# Micro-Professor™
# MPF-I Student Work Book

MULTITECH INDUSTRIAL CORPORATION

Copyright © 1982 by Multitech Electronics Inc.
and Multitech Industrial Corp.
No part of this publication may be reproduced,
stored in a retrieval system,
or transmitted,
in any form or by any means,
electronic, mechanical, photocopying,
recording, or otherwise,
without the prior written permission of the publisher

MULTITECH INDUSTRIAL CORPORATION
OFFICE/ 977 MIN SHEN E. ROAD, TAIPEI, 105,
TAIWAN, R.O.C.
TEL:(02)769-1225(10 LINES)
TLX:23756 MULTIIC, 19162 MULTIIC
FACTORY/5. TECHNOLOGY ROAD III,
HSINCHU SCIENCE-BASED INDUSTRIAL PARK
HSINCHU, TAIWAN, 300, R.O.C.
TEL:(035)775102(3 LINES)

Multitech Electronics Inc.
195 West El Camino Real
Sunnyvale, CA. 94086
U.S.A.
Tel: 408-7738400
Tlx: 176004 MAC SUVL
Fax: 408-7498032

# Micro-Professor™
# MPF-I Student Work Book

## CHAPTER 1.


1 • 1 Unpacking and Installation ........ 4
1 • 2 Programming Languages ........ 5
1 • 3 Testing & Familiarization ........ 5
1 • 4 Program in English ........ 6
1 • 5 Program Explained ........ 6
1 • 6 Assembly Listing ........ 7
1 • 7 Checking for Data Entry Errors ........ 8
1 • 8 Program Execution ........ 9
1 • 9 Checking the Results ........ 9


## CHAPTER 2.


2 • 1 Reset or Monitor: What's the Difference? ........ 13
2 • 2 Is the MPF-I a New Recording Artist? ........ 15
2 • 3 More Keys ........ 17


## CHAPTER 3.


3 • 1 ASSEMBLY--the Sane Way to Go ........ 21
3 • 2 Easier to Read ........ 22
3 • 3 Easier to Program ........ 23
3 • 4 Easier to Correct ........ 23
3 • 5 How to Proceed Using the MPF-I ........ 24


## CHAPTER 4.


4 • 1 Central Processing Unit (CPU) ........ 29
4 • 2 PIN-OUT ........ 30
4 • 3 Memory ........ 31
4 • 4 RAM ........ 32
4 • 5 Dynamic RAM, Static RAM ........ 33
4 • 6 ROM ........ 33
4 • 7 Monitor Program and ROM of the MPF-I ........ 34

4 • 8 Address ... 35
4 • 9 Byte, Bit ... 36
4 • 10 Clock ... 37
4 • 11 Reset ... 38
4 • 12 Ports ... 38
4 • 13 Peripherals ... 38
4 • 14 Parallel I/O Lines ... 39
4 • 15 Advanced Hardware Description (Optional) ... 41
4 • 16 Power Supply ... 42


## CHAPTER 5


5 • 1 Learn by Doing ... 51
5 • 2 Flashing a Message ... 51
5 • 3 Program Analysis ... 52
Memory Checking ... 72
EPROM Testing ... 72
CPI ... 77
RAM ... 81
Questions ... 82
Answers ... 90


## CHAPTER 6


6 • 1 Exercises and Experiments ... 97
6 • 2 Questions of Exercises ... 123
6 • 3 Answers to Exercises ... 133


## CHAPTER 7


7 • 1 Major Divisions of the Monitor ... 168
7 • 2 The Code ... 169
Answers to Exercises ... 185

## CHAPTER 8


Sheet 1 of 4 ........ 189
Sheet 2--The Control Function of the 8255 ........ 203
Sheet 3--Counter Timer Circuit (CTC) and Parallel I/O (PIO) ........ 204
Sheet 4 ........ 204


## APPENDIX


Appendix A References ........ 205
Appendix B Alphabetical Listing of Monitor and Interrupt Key ........ 208
Appendix C ........ 210

# CHAPTER 1

## Introduction to MPF-I

This workbook is designed for the first time user of microprocessors and microcomputers but intends to explore the world of microcomputers. The workbook guides you step by step in your learning about microcomputers. We know that you will learn a great deal and also enjoy becoming familiar with microprocessors.

The fastest and most pleasant way to learn is to learn by doing. You are encouraged to use a MPF-I microcomputer to do the interesting experiments so that you can learn more quickly.

This workbook will first teach you to press a few keys on the MPF-I to see how it responds. And then, the workbook will teach you to press more keys and let the MPF-I show you the interesting results. As you progress in this workbook you learn new modes of operation. What is more important, you will eventually learn a great deal about microcomputers and microprocessors. To put it simply, you will know how to use computers to solve problems.

Never let a computer scare you! When automobile was first introduced to the world, few people were familiar with it. Even today, you don't have to know everything about an automobile to drive it. For example, you don't have to know too much about the complicated automobile transmission system to drive a car. But of course, you have to know some basic principles so that you can shift the gears properly. Operating a computer can be reduced to basic principles. Once these principles are learned, you can determine whether you want to continue and become a customer engineer (auto mechanic), an operator (a professional driver), or a designer (an automative engineer.)

To learn how to drive a car, you must become familiar with the features or functions of some devices or equipment such as the engine, steering wheel, etc. (In the realm of computer, these devices or equipment are generally referred to as "hardware".) You must at least know the names of some computer hardware devices and equipment and their basic functions. Once you have learned to drive a car, your every move comes naturally and easily. The same is true about (operating) a computer.

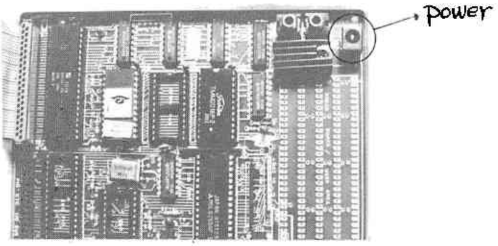
The manuals that accompany your Microprofessor are designed for reference and to suggest experiments by showing examples. To get started, it is suggested that you follow the procedures given below.

**Exercises and Experiments.**

As you proceed through this workbook, you will see the notation Exercise 6-1, Exercise 6-2,..., in the left margin. This is a signal to proceed to the section named EXERCISE and find the same number 6-1, 6-2,.... You should answer any questions in the exercise and then proceed to the ANSWER section to check your work. You will also be asked to perform experiments (answer questions) in the Experiment Manual (Hardware/Software). The answers to these questions are usually found in the section named EXPERIMENTS. Occasionally, an answer to an experiment will be part of an answer to an exercise.

## 1 · 1 Unpacking and Installation

Open the "book" containing the Microprofessor (MPF-I). Locate the power connector in the upper right-hand corner. (Fig. 1-1)



Find the AC adaptor. The adaptor (Fig. 1-2) is a black box labeled "AC ADAPTOR MULTITECH". You should make certain that the voltage input shown on the adaptor matches the voltage supplied by your outlet. In the United States it is assumed (unless a special order is made) that the supply is 117 VAC - which is usually referred to as one-ten (110 V). You should also check the frequency; the label on the adaptor will show the frequency in hertz (Hz).

Plug the 9V circular shaft into the power receptacle on the MPF-I. The side opposite the AC adaptor label is to be plugged into your AC power outlet.

```
****************************************************
* CAUTION : DO NOT TOUCH THE PRONGS WHILE PLUGGING *
* THE AC ADAPTOR INTO YOUR OUTLET!                 *
****************************************************
```

## 1 • 2 Programming Languages

What is a program? How can a program be run (executed)? To answer these questions, you should know how a computer communicates with the people who use it. A computer sometimes can be regarded as a loyal servant who always follows the instructions given by the master. Once the master has some good tasks for a computer to do or requires a computer system to solve some problems, the master gives step-by-step instructions to the computer. Each and every step that is required to solve a problem or to perform a task are given clearly to the computer. These instructions constitute a program. Any person who writes a computer program is called a programmer. In order to program, you have to learn computer programming languages such as ASSEMBLY, BASIC, PASCAL, APL, FORTRAN, and FORTH. We will discuss ASSEMBLY language in later chapters.

Now you know that a programmer can give instructions to a computer. How does a computer talk to a person? In the case of the MPF-I, a six digit LED (light emitting diode) display and a built-in speaker are used to tell a programmer what the MPF-I is doing. The MPF-I display can show modified Roman letters and Arabic numerals from 0 to 9 plus some special signs.

## 1 • 3 Testing & Familiarization

In the exercise below, you will be shown how to enter and execute a short program. Performing this exercise will test some of the MPF-I functions and familiarize you with the MPF-1's Z80 microprocessor. The program used in this chapter adds two numbers, and stores the result in memory.

## 1 • 4 Program in English

Load the first number (5) into the A register, and the second number (4) into the B register. Add the content of the B register (4) to the content of the A register (5), and put the result (9) in the A register. Then, store the value of the A register in memory location 1830H (H stands for hexadecimal) and finally halt the Microprofessor.

If you are already familiar with registers and ASSEMBLY language programming, you may want to skip the next section, although it is highly recommended for anyone.

## 1 • 5 Program Explained

In the program, you will instruct the MPF-I to access the A register and load it with a value : (5). Now you may ask : "What is a register?" A register is an area in the CPU that stores different kinds of information. It can be regarded as a memory and a work area. Generally, the registers of Z80 CPU are divided into two categories--general purpose registers and special purpose registers. The general purpose registers are named A, B, C, D, E, F, H, and L. The special purpose registers include PC, SP, IX, IY, I and R. In the case of our program, 5 is placed in the A (accumulator) register. Because the A register must contain one of the values in any 8-bit arithmetic operations. It is, therefore, often called the Accumulator. When 5 has been loaded into A, 4 will then be loaded into B register. The values in the A and B registers will be added together and placed into the A register. The value in the A register will be stored at memory location 1830H, then the MPF-I will be halted.

## 1 · 6 Assembly Listing

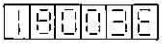
All of the program is entered into the MPF-I in hexadecimal (hereafter, we will use the common abbreviation hex for hexadecimal.) Therefore, you first write your program in Assembly language and then translate it into hexadecimal. Most of the demonstration programs written in MPF-I manuals will also be listed in machine language code which is in hexadecimal. A complete Assembly program listing is shown below.

```
1800   3E05     LD     A, 5
1802   0604     LD     B, 4
1804   80       ADD    A, B
1805   323018   LD     1830, A
1808   76       HALT
```

You will now enter the object (machine) language code shown in the Assembly, program listing. If you haven't already done so, connect your MPF-I to the power source. Now press the system reset key [RS](the [RS] key is used for initializing the MPF-I). Since the memory locations at which you can store programs begin at hexadecimal location 1800H, entry of object code will start at 1800H. Press the address key [ADDR] . A random address will be displayed on the four leftmost digits; these digits will be referred to as the address field.

Enter the starting address for the machine language code by pressing [1] , [8] , [0] , [0] . The same result can be obtained by pressing the program counter [PC] key (this only works when your program starts at 1800H). Now inform the Micro-Professor that data is to be entered by pressing [DATA]. Refer to line 2 of the assembly program listing. Line 2 contains two bytes of object code 3E and 05.

Key in the first byte by pressing [3] and then [E] . The display should now show:




Advance the address field display by pressing [+] . The display will show:

Enter the second byte of hexadecimal data by pressing Ø and then 5 . The display should be:

180105

Line 3 of the listing also contains two bytes of hexadecimal data; enter these bytes by keying:

[+], [Ø], [6], [+], [Ø], [4],

In a similar manner, enter the rest of the program, namely:

[+], [8], [Ø], [+], [3], [2], [+], [3], [Ø], [+], [1], [8], [+], [7], [6],

## 1 • 7 Checking for Data Entry Errors

The program has been entered. It is wise to check for entry errors. Press [ADDR], [1], [8], [Ø], [Ø]. Are the right-most two digits in the data field equal to 3E? If not, press [DATA] and enter [3], [E]. To examine the next byte press [+]. Is there a Ø5 in the data field? If the display is correct continue inspection of all the remaining data using the [+] key. If the present byte or any successive bytes are incorrect, enter the correct data.

## 1 • 8 Program Execution

There are two ways to begin execution at address 1800H. The simplest is to press [RS], [PC] and then [GO]. (The [PC] and [GO] keys are used in program execution. [PC] stands for program counter. This key is used to tell the MPF-I where a program begins. The [GO] key is a signal (that says: "You may go execute the program".) The second method allows execution to begin at any address. Press [RS] , [ADDR] , the beginning execution address e.g. 1 , 8 , 0 , 0 , then press [GO]. When you press [GO] (in the above program), the screen will go blank and stay blank. The program has reached the HALT instruction and is waiting for the next operator action.

## 1 • 9 Checking the Results

To regain control of the keyboard functions, press MONI. The answer to 5+4 was stored at location 1830H. Key in [ADDR] , [1] , [8] , [3] , [0] . The display should show:

Now let's check what was stored in the registers. Press the [REG] key. The word REG should show on the display.

Press [AF]; this will display the contents of the AF register pair. The first two digits contain the contents of A register, and the middle 2 digits display the contents of F register. Do not worry about the F register now. We are only concerned with the value in A register. Didn't we store a five in A register? And then, didn't we add the contents of B register (4) to the contents of A register? If A register contains a nine, then it is correct. Press [REG] then the [BC] key. In this case we are looking at the contents of the BC register pair. Are the numbers in the leftmost 2 digits 04? If they are, then Congratulations! You have just successfully entered your first object code program onto the MPF-I. If something went wrong, you may find the answer to your problem in the next section.

When you made the following errors:

1) A byte was incorrectly entered. Write the correct byte over the incorrect byte.

2) One or more bytes were left out. Read section 3.3.3 (in the User's Manual), then remove the bytes one by one.

3) One or more bytes need to be added. Read section 3.3.2 (in the User's Manual), then add each byte.

# CHAPTER 2

## Keyboard Familiarization

Are you Keybored? O.K. Now you know how to enter a program and, so far, your experience with running a program has been successful. But if you're like us, you may be KEYBORED!
Some symptoms of this disease are confusion with each key functions, and adversion to abbreviations such as ADDR, REG, SBR and INS, and finally, allergic reactions to white, grey and orange rectangles. The good news is that this disease is painlessly curable, our RX: read this chapter and find out how to avoid entering the same program over and over.

## 2 • 1 Reset or Monitor: What's the Difference?

In chapter 1, you entered a program, and you were told there were two ways to stop the execution of a program. One was to press the MONI key, in which case the display shows a memory address, or you could press the RS key, in which case the display will show µPF-I. If you were sharp, you might have noticed that we didn't press the RS key to stop the program when we were planning to look at the contents of the registers. This is because of the RS key is used: (1) to perform a hardware reset of the CPU, (2) to initialize the monitor program, and (3) to transfer control to the monitor.

If we were to initialize the monitor program before we went to check the values in a register, those values might not remain the same. Unlike the RS key, the MONI key transfers control immediately to the monitor. The address at which the program was currently at when the MONI key was pressed is displayed along with the data at that location. Enter the following program, and we'll do a short experiment with the MONI and RS keys.

```
                      ;police car siren:
1800                          ORG     1800H
1800  0E00              LOOP  LD      C,0
1802  21C000                  LD      HL,0C0H
1805  CDE405                  CALL    TONE
1808  0EC0                    LD      C,0C0H
180A  210001                  LD      HL,100H
180D  CDE405                  CALL    TONE
1810  18EE                    JR      LOOP
                              ;
                        TONE  EQU     05E4
                              END
```

If you had any problems entering the above program, you need to review chapter 1.

Now for the experiment, after you have loaded the program, press PC, then GO. If everything was entered correctly, you should hear a sound similar to a European police car siren. Now, to stop the execution, press RS . What happens? µPF-I appears on the display. Begin the program again and this time, stop it with MONI. What happens this time? Instead of going back to ground zero and initalizing the system, MONI simply halted the program

where it was and allowed you to examine the registers. When the program is halted the left 4 display digits show the program counter (where the program was halted) and the 2 right display digits show the opcode at the halted address. Press GO and MONI several times and notice that the contents of the PC counter address will vary.

## 2 · 2 Is the MPF-I a new recording artist?

Well, not exactly. But the MPF-I does make tapes. Examine the top, righthand corner of your MPF-I. Next to the power socket, you will find two circular metal jacks. When a cassette recorder cable is connected to these sockets and to a recorder, a simple storage of data can be performed. Assuming you have the required cable and recorder, let's make a tape of the police siren program you have use just entered. You may wish to check to see if the siren program is still in the MPF-I memory. If not, reenter the program. Connect the cables from the cassette recorder to your MPF-I. Make sure to connect the cable from the EAR jack to the MPF-I's EAR socket. Do the same with the MIC cable and socket. Now, press on the [TAPE WR] key. The screen will show a random number in the address field. The display should be similar to this :

| x.x.x.x. -F |
|---|

The -F in the data field is the mnemonic for (stands for) filename. The filename is used to distinguish different data sets stored on a single cassette. It is also used to read back data. You can use any combination of the 16 lettered and numbered keys in the filename. For your first try, let's use something easy to remember, e.g. ØØØ1. Enter Ø,Ø,Ø,1. Now enter [+] to move on to the next display. You will again see a random number in the address field and the display should look like this:

| x.x.x.x. -S |
|---|

The -S in the data field stands for the starting address of the data you wish to put on the tape. Our program begins at 18ØØ. Enter 1,8,Ø,Ø. Now press [+] to get to the next field again. You should see a random number, then the mnemonic on the display should read -E. This signifies that the last memory address to be written to the tape should be entered. The last address in our program was 1811 so enter [1], [8],[1],[1]. Now we are ready to make a tape. Rewind the casette in your recorder to the beginning of the tape. Press PLAY and RECORD on the recorder, then press [GO] on the MPF-I. If everything is going correctly, you should be able to hear the noise of data being output. What sounds noisy to you is actually your program! If the cassette recorder is not ready and you press [GO] , do not worry, the MPF-I will still send out data and then return control to the user. You can then begin the process over again.

Now let's read the data we wrote to tape back into the MPF-I. Press [TAPE RD]. We now have that familiar mnemonic (-S) on the screen again. Input 0001, or whatever filename you used in the above exercise. Rewind the cassette and press [GO] on the MPF-I. Press [PLAY] on the recorder. The screen will go blank, periods will be displayed for a few moments, now the filename of the program at that location will be displayed. In this case 0001, the program will now be read in. When the "noise" stops, stop the recorder and reset the MPF-I. Now press [PC] and [GO] . Is the program the same? If so, congratulations! If not repeat the above process with a different volume setting.

## 2 · 3 More Keys

The MPF-I allows users a great deal of flexibility and power through keyboard entries. How does a user become familiar with the keyboard functions? An appendix with an alphabetic listing of the keys is at the back of this manual. But, do you really need to read about each key? I recommend you proceed through the manual and learn how to use the keys in the context of programming. Use the appendix for reference.

### Keyboard Familiarization Questions

1. Which keys do not cause a tone to sound when pressed?

2. Why is the RESET key the only key that is brightly colored?

3. Look at the MPF-I User's Manual, Table of Contents-3, Operation introduction. Two of the gray keys are not listed -- which ones?

4. Can you press any key that would cause damage to the MPF-I?

5. There is a magic key that will tell the Micro-Professor I to do exactly what you want done. What is this key?

### Keyboard Familiarization Answers

1. RESET, MONI, INTR, USER KEY.

2. This key is the MPF-I PANIC button. The color should also serve as a warning that the current contents of the registers will be lost, when RESET is pressed.

3. INTR and USER KEY. Additional programming must be done to make these keys perform a function.

4. No, not unless you hit the key with a hammer. Pressing the wrong key can change your program.

5. GO. If a program has been entered and it is correct.

# CHAPTER 3

## Keeping Your Sanity—
## (or How not to Write in Object Code)

## 3 · 1 ASSEMBLY--the Sane Way to Go

In earlier chapters there have been hints that you should first write your program in Assembly Language. The major reasons for Assembly programming are:

- .Easier to read
- .Easier to write programs
- .Easier to correct errors

## 3・2 Easier to Read

What does the 3 instruction program below do?

```
0011 1010 0000 0000 0001 1010
1100 0110 0000 1000
0011 0010 0000 0010 0001 1010
```

After looking at the binary code you probably don't care. OK! Here is the same program in hexadecimal.

```
3A 00 1A
C6 08
32 02 1A
```

How can the hexadecimal program be decoded? Open the MPF-I User's Manual to Appendix C. Find the section Z80-CPU INSTRUCTIONS SORTED BY OP-CODE. Search for the opcode 3A (Second column almost halfway down). The row reads

```
3A 8405      LD A,(NN)
```

OH! LD stands for load.
A load means making a copy of the data, usually one or two bytes, then entering the data into a stated destination. In this instruction, a byte is loaded from memory into A register. The form LD A,(NN) is still a little hard to read. The Assembly language instruction is

```
LD A,(1A00H)
```

which means :

(1) find memory location 1A00 (hexadecimal),
(2) make a copy of the byte at location 1A00,
(3) then replace contents of the A register with the copy from memory.

The entire program is

```
LD A,(1A00H)   ; A <— (1A00H)
ADD A,8        ; A <— A + 8
LD (1A02H),A   ; (1A02H) <— A
```

This program :

1) loads a value from memory into A,
2) adds 8 to the contents of A,
3) puts the result (A register) in memory location 1A02H. Read the binary code again and compare with the Assembly language program.

## 3 • 3 Easier to Program

In your program a test is to be made. If the value in the A register is zero, then a routine which clears the account book is to be executed. If the value of A is negative, then an overdraw routine is executed. Using Assembly language you can write:

```
        JP    Z,CLRACC     ;If  A=0 jump to clear account
        JP    M,OVERDR     ;If  A is minus ( negative) jump
                                  to overdrawn.
```

In object code programming (hexadecimal or binary) you may not know where the routines CLRACC and OVERDR will be in memory. This means you will have to leave a blank area in the code. Too many blank areas lead to the inability of locating the exact address where the jump was to be made to. In assembly language programming you just write the name of the routine e.g. CLRACE.

## 3 • 4 Easier to Correct

Sooner or later it will become necessary to alter codes--insert, delete, or add instructions. In Assembly language programming, you can usually find the code to be modified swiftly. To add a new line,simply write the instruction in mnemonic form.

## 3・5 How to Proceed Using the MPF-I

1. Decide what the program must do. Base your decisions upon the required input and output.
2. Decide if you can write the program. You might be asked to compute an advanced mathematical function of which you have no knowledge.
3. Decide whether the MPF-I can program the task. Unless a special interface is designed; electro-cardiograms can't be read directly.
4. Organize the program flow. Sometimes a flowchart helps.
5. Write the program in Assembly Language.
6. Hand translate the program into object (hexadecimal) code.
7. Enter the hexadecimal code into the MPF-I's memory.
8. Test the program.
9. Make corrections in Assembly language and translate into object code.
10. Save the working program on tape.

## QUESTIONS

1. Turn to Appendix C in the MPF-I USER's MANUAL. Find the section Z80-CPU INSTRUCTIONS SORTED BY MNEMONIC. The table should begin with:

| OBJ CODE | SOURCE STATEMENT |
|---|---|
| 8E | ADC A,(HL) |

Use the table in the manual to fill in the missing entries in the table below.

| OBJ CODE | SOURCE STATEMENT |
|---|---|
| | ADD A,(HL) |
| | CCF |
| | NEG |
| | LD (DE),A |
| | XOR N |
| | BIT 3,H |
| | SRA A |

2. In this section you will be asked to translate from object code (written in hexadecimal) to source code (written in assembly language). This is usually done when you can't read the source statement or are given some code in hexadecimal (this is a rotten situation).

Turn to Appendix C in the MPF-I User's Manual. Find the section Z80-CPU INSTRUCTIONS SORTED BY OP-CODE. The table should begin

| OBJ CODE | SOURCE STATEMENT |
|---|---|
| 00 | NOP |

Use the table in Appendix C to fill in the entries in the table below:

| OBJ CODE | SOURCE STATEMENT |
|---|---|
| 70 | LD (HL), B |
| FF | RST 38 |
| 00 | NOP |
| 50 | LD D, B |
| A6 | AND (HL) |
| CB 10 | |
| DD CB05CE | |
| ED B0 | |
| FD 23 | |

Until you looked for CB, all you had to do is to find the object code is to go down a list in hexadecimal order - 0, 1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. All instructions starting with CB,DD,ED, and FD are in separate lists. The reason for the separate lists is that the Z80 executes these instructions differently. In a later chapter, some of these instructions will be explained.

**ANSWERS**

1.

| OBJ CODE | SOURCE STATEMENT |
|---|---|
| 8E | ADC A,HL |
| 3F | CCF |
| ED 44 | NEG |
| 12 | LD (DE),A |
| EE 2Ø | XOR N |
| CB 5C | BIT 3,H |
| CB 2F | SRA A |

2.

| OBJ CODE | SOURCE STATEMENT |
|---|---|
| 7Ø | LD (HL), B |
| FF | RST 38H |
| ØØ | NOP |
| 5Ø | LD D,B |
| A6 | AND (HL) |
| CB 1Ø | RL B |
| DD CB05CE | SET 1,(IX+D) |
| ED BO | LDIR |
| FD 23 | INC IY |

# CHAPTER 4

## Introduction to Hardware

This chapter will introduce to you some of the basic components (by basic, we mean they are indespensable.) and their functions.

Computers have been called "electronic brains" because computers can perform such operations as logic comparisons, arithmetic calculations, and more recently reasoning. But computers are much more than an electronic brain. Computer have become more like an individual human being. This will be discussed later.

## 4 • 1 Central Processing Unit (CPU)

The "brain" of a computer or a microcomputer is its central processing unit (CPU). You may wish to know what a CPU looks like. The MPF-I has a Z80 microprocessor which is used as a CPU.

You can locate the Z80 CPU of MPF-I in a diagram on page 4 in the MPF-I User's Manual. At the upper left corner of the diagram, there is an rectangular area marked with Z80 CPU. Here is where the CPU is located.

You may have noticed that there is a notch on the upper edge of the Z80 CPU. The notch is used to indicate whether the Z80 CPU is inserted correctly into the socket. If the notch points upwards, the Z80 is correctly inserted. Otherwise, the Z80 CPU is not adaquately inserted and the MPF-I would run into trouble. Typically, reverse insertion causes the Z80 to overheat until it burns up.

Why does the Z80 CPU have to be mounted correctly? To answer the question, let's take a look at the CPU. The CPU is an n integrated circuit (IC) chip which is a tiny piece of silicon on which many microscopic circuits are built. The chip is packaged in two pieces of a Dual-In-Line package (DIPs) that keeps moisture, dust, and impurities away from the chip. But since the chip is sealed in the DIP package, the circuits inside the package need to be connected to outside circuits through pins as shown in the diagram on page C-1 in the Appendix C of the MPF-I User's Manual.

## 4·2 PIN-OUT

To make sure that a circuit inside the package is connected to a circuit outside of the package collectly, a specific pin is assigned to make a correct connection. As a result, each pin is given a specific pin number.

The diagram on page C-1 shows how pin numbers are assigned to pins. If an IC chip is inserted in reverse (that means the notch of the chip points downwards.), it results in incorrect connections of circuits. The pins are not numbered sequentially (1,2,3,4,...) but rather by function. For example the transfer of data in and out of the Z80 CPU is accomplished thru 8 data pins (14, 15, 12, 8, 7, 9, 10, 13). These 8 pins are grouped together and called the data bus.

There are several reasons for selecting Z80 as the CPU for the MPF-I. First, Z80 is one of the most popular microprocessors. It is used as the CPU of many microcomputers. Many software programs have been written to run on Z80 based computers. You can share or exchange software programs with others. Secondly, the Z80 instruction set was designed as an extention of the instruction set for the Intel 8080 microprocessor. Therefore, almost any program written for an 8080 microprocessor can be executed on a Z80 microprocessor without any changes. The 8080 microprocessor is a very important microprocessor chip, for which many software programs already exist. Thirdly, the Z80 microprocessor (Z80 CPU) features two sets of general-purpose registers and additional special purpose registers which make it easier for computer users or programmers to write programs for Z80 based microcomputers.

## 4 • 3 Memory

Before we proceed to show how a CPU interacts with other devices, let's take a look at one of the major parts that constitutes a computer--memory. A human being must have a memory so that he or she can learn and think. A computer must have a memory in order to process information and solve problems.

Memory is generally defined as any device that can store data in such a manner that the information can be accessed (or reached) and retrieved (or fetched). In today's computers, the memories usually come in the form of IC chips. The appearance of these chips look similar to that of a CPU such as the Z80 microprocessor. They have DIP packages and pins. Each chip is assigned a specific number. This number indicates the functions the chip can perform.

## 4.4 RAM

Now open the cover of your MPF-I, there is a 24-pin IC chip on the upper right part of your MPF-I. On page I-4 of your manual the chip is labeled RAM. The chip which is marked with either 2016, 58725, or 6116P-4, is a 16K static random access memory (RAM). On the part of the printed circuit board just above the IC memory chip, the words "U8" is marked to identify the location where the chip should be installed.

When you try to decipher the words "RAM" and "static" you may become frustrated. These words are just used to distinguish different types of memory chips. The most commonly used types of memory are RAMs, ROMs (read only memory), and EPROMs (erasable programmable read only memory).

The RAM, more correctly speaking, should be referred to as read/write memory. A more correct definition of RAM is random read/write memory. The RAM is a semiconductor memory into which information (data) can be stored (written) and retrieved (read out) again. RAMs differ from ROMs-- once the power supply of a computer is turned off, the contents of a RAM disappear. As a result, RAMs are suitable for storing data which are to be used temporarily by a computer such as programs and data.

## 4・5 Dynamic RAM, Static RAM

The RAM can be further divided into two types--static RAM and dynamic RAM. The static RAM is what is generally referred as those RAMs whose contents disappear, will only change, when written into or as soon as the power supply is turned off. The dynamic RAMs, even when power is continuously supplied can lose data if the contents of such RAMs do not go through a memory refresh process. Unlike some (many) CPUs the Z8Ø provides a refresh signal.

## 4・6 ROM

Data is read from a ROM. No data can be written into ROM chips. Even when the power supply is cut off, the contents of ROMs do not change. ROM chips are suitable for storing data that is to be used repeatedly.

## 4 • 7 Monitor Program and ROM of the MPF-I

The location indicated by U6 is used to put a ROM for storing monitor programs. Almost every microcomputer uses a ROM or an EPROM memory chip for storing monitor programs, which are used to control the internal operations of a microcomputer. An EPROM is a close relative of the ROM. By applying ultraviolet rays an EPROM can be erased.
Typical functions of a monitor program include the initialization of the CPU, keyboard scanning, display control, and responding to the function to be performed each time a key on the keyboard is pressed. In short, once a microcomputer is turned on, the CPU of the microcomputer begins to execute a monitor program. At location U6 in the MPF-I, either 16K PROMs such as 2716 and 2516 or 32K PROMs such as 2732 and 2532 can be used for storing monitor programs.

We have talked about the CPU, memory, and data input device (such as the MPF-I keyboard), and data output device (the display and speaker). Most of today's microcomputers have these four major components.

## 4 • 8 Address

Just by watching the keyboard, you may guess that a programmer can key in a character like "A" or "7". But where can a character like "A" be stored in the MPF-I. How is it stored? A computer is designed so that it only recognizes "Ø"s and "1"s no matter who the manufactuer is. As a result, when you press a key to store a word, the computer first encodes the word into the series such as Ø11Ø1ØØ1 and then stores the string of Ø's and 1's into a specific location. Since the computer memory stores vast amounts of data, data should be stored or retrieved from specific locations to avoid confusion in data manipulation. Therefore, an "address" is given to identify the location of a specific item of data the same way as a specific building is assigned an address so that mail addressed to the building can be delivered properly.

ADDRESS BUS

The Z8Ø microprocessor uses 16-digit binary numbers to identify the locations of data stored in the memory devices that are connected to it. When the CPU of a computer intends to access the data stored in its memory devices, it communicates with its memory through a 16-line address bus. Each line of the address bus corresponds to a binary digit of the 16-digit address. And each line of the address bus can convey two signals to the memory--"Ø" and "1". Using Ø and 1, you can construct 65,536 16-digit numbers. That means the Z8Ø CPU can access up to 65,536 memory locations. The number 65,536 is often written 64K.

## 4 • 9 Byte, Bit

We have mentioned that data is stored in the form of strings of Ø's and 1's in a computer. In computer systems, memory size is measured in bytes. In Z8Ø based microcomputers such as MPF-I, a byte is equal to eight binary digits, e.g. 1. A byte looks like ØØØØØØØØ, 11111111, 11ØØØ1Ø1, or Ø1111ØØ1. A byte is made up of eight "bits". In a binary numerical system, a bit is either a "Ø" or a "1".

You may wonder how an item of information or data is accessed (for example from the keyboard). Turn to page I-B-3 (sheet 2 of 4). This schematic shows how the IC (8255) controls the input and output of data of the MPF-I. If you have not worked with hardware, do not expect to understand the details of how the 8255 controls devices such as the displays. Later in the workbook a detailed explanation will be given of the schematics. This chip controls MPF-I's data input and output devices such as LED displays, the keyboard, the cassette interface, the interface to MPF-I's CPU, and the address decoder. In the lower left part of the schematic (A, 7 and 8), you will find a chip (74LS139) which is connected to a pin of the 8225 chip marked CS (which stands for chip select). The 74LS139 is an address decoder used for deciding what range of memory addresses is being accessed by the CPU. There is a "--" on top of the mark CS. That means the address decoder works when the input of CS is low. A low means the voltage is pretty close to zero - probably Ø.4 volts. We say the address decoder works is active low, because when the input of CS is low it becomes active.

## 4 • 10 Clock

Chips (or large-scale integrated circuits, LSIs) in the Z80 family require a clock. The clock supplies a square wave of a certain frequency used for controlling transfer of data in the CPU. Every time the clock ticks, data is tansferred. The illustration below shows how a square wave looks like.

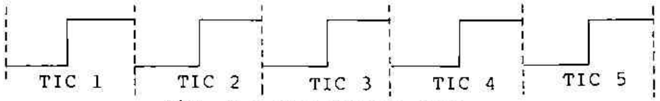



Fig 4-1 The square wave

Chips using a clock have specific requirements for the High and Low voltages. A good source for a clock is a crystal oscillator. On a schematic, it looks like fig. 4-2.

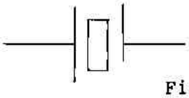


Fig 4-2 crystal oscillator

On sheet 1 of 4 of the MPF-I schematic, you can locate the crystal oscillator at (D-7) and (D-8).

The output of the crystal oscillator is connected to pin 3 of the IC 74LS74 (coordinates D-6), and then to pin 6 of the Z80 CPU (D-5). The standard designation for a clock is $\Phi$ . The label $\Phi$ is the point where clock signals go into the CPU.

## 4 • 11 Reset

A requirement for a circuit to work properly is that it always starts the same way each time it is put to work. The Z80 CPU always starts (comes up) by addressing location 0000 when power is supplied and a pin called RESET is held low for a few cycles. Any time your MPF-I appears to be out of control, you may activate a circuit that resets the CPU. Pressing the RS button controls the circuit that supplies a reset signal to the Z80 CPU.

## 4 • 12 Ports

Now we will take a closer look at the schematic for MPF-I input and output (sheet 2 of 4). On the right side of the 8255, there are three "ports". You may ask how ports can be built on a tiny 40-pin chip.

The word port conventionally means a harbor, a sea port where ships can sail in or out, loading or discharging large amounts of goods. In our study of microprocessors, a port can be regarded as a place where data from outside can be "loaded" into the CPU and where a CPU can "discharge" the data it has processed.

## 4 • 13 Peripherals

The chip 8255 is a 40-pin programmable peripheral interface IC. Peripherals are generally referred to as those devices which interact with the CPU for certain purposes. If you use a cassette tape recorder to record data or programs, then we say the cassette tape recorder is a peripheral of the MPF-I. Peripherals can be a printer, auxiliary memory storage equipment, or a display terminal, etc.

## 4 • 14 Parallel I/O Lines

Of the 8255's 40 pins, there are 24 pins used as parallel input/output lines (we will use I/O instead of input/out hereafter.) The word parallel may puzzle you.

When data is transferred bit by bit, we generally call this method a serial data transfer. Data is transferred over telephone lines serially. If you want to input or output eight bits of data or several batches of data all at once, you have to use parallel I/O lines. In computer systems, data is usually transferred byte by byte between the CPU and ROM or RAM chips. As a consequence, we have to use parallel lines to connect the CPU and its memory devices. If a byte--01001001--is fetched by the CPU from its memory, each bit of this byte will be carried by a single parallel line to the CPU. Therefore, a data bus consisting of eight parallel bi-directional lines is used to supply data between the CPU, memory, and I/O ports.

The 24 parallel I/O lines of the 8255 are divided into three ports--Port A, Port B, and Port C--with each port having eight parallel I/O lines. Each of the three ports is called an 8-bit port. Port A is an input port, because this port is used for collecting data (which will then be transferred) to the CPU. Port B and C are output ports, because the two ports are used for activating displays and keys.

You can locate Port A on the schematic sheet 2 of 4. In the lower right part of the IC 8255, there are eight pins marked with PA0, PA1, PA2,...PA7. They are connected to eight parallel lines. Pin 37 (the pin marked PA7) is used for inputing data stored on cassette tape into the MPF-I. Pin 38 (the pin marked with PA6) is connected to the User key, which will become active when the signal on it is low. PA0 through PA5 are connected to six rows of the keyboard matrix. The input signal becomes low only when keys in the active column are pressed. Since the 8255 is programmable, a programmer can program a port to be input or output.

In the MPF-I, Port B is an output port used for controlling the LED displays. As you can see on the schematic, PB0 through PB7 is wired to the displays with eight parallel lines. Each pin or bit of Port B is used to control one of the seven segments of the LED display and the decimal point. Fig.4-3 shows the name of each segment and the corresponding bit in Port B.

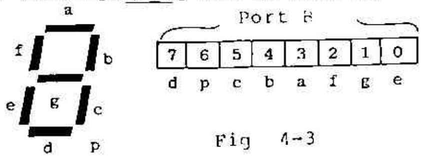



Fig 4-3

Port C has many functions. Bit 7 of Port C (PC7) is used for writing data into cassette tape. It is also connected to the speaker and an tiny LED lamp. Once you press a key on the keyboard of the MPF-I, the speaker of the MPF-I will generate a sound and the LED lamp will blink. Except for the keys marked with [RS], [MONI], [INTR], and [USER], all the other keys cause the LED lamp to blink and the speaker to generate a sound.

The PC6 is used for single step execution of a program or when break points exist in a program. Bit Ø through bit 5 are connected to the LED displays and the keyboard matrix. Bit Ø selects the rightmost LED display and bit 5 selects the leftmost LED display. All these bits are active high.

Thus PCØ through PC5 are used for selecting LED display. For example, when PCØ is high, the rightmost display of the LED displays is active.

You may have noticed that the parallel lines of Port B and C first go through three blocks marked with 75492. The three blocks are actually three ICs used as drivers that amplify the incoming signals and convert them into strong signals.

When you use a cassette tape recorder to read data to the MPF-I CPU, the data goes into the CPU through PA7. When the CPU of MPF-I writes data into a cassette tape, the data goes to the cassette tape through PC7.

## 4 • 15 Advanced Hardware Description (Optional)

### 4 • 15 • 1 PIO: Parallel I/O Circuit

The Z80 parallel I/O circuit (PIO) is one of a set of chips manufactured to facilitate Z80 interfacing. The PIO circuit is designed to provide a two-port, programmable, TTL compatible parallel data transfer between the Z80 CPU and peripheral devices. Turn to schematic sheet 3 of 4. In the D and C of column 4, you can find Port A and Port B. The two ports are independent 8-bit parallel bi-directional peripheral interface ports using "handshake" data transfer method.

The Z80 PIO is an IC chip with 40 pins. Of the 40 pins, D0 through D7 is used as Z80 CPU data bus. This is a bidirectional, tristate bus which is used to transfer all data and commands between the CPU and PIO.

### 4 • 15 • 2 CTC: Counter-Timer Circuit

The Z80 counter-timer circuit, like the Z80 PIO circuit is one of a group of IC chips manufactured to facilitate Z80 CPU interfacing. This chip performs timing and event counting functions with four independent 8-bit channels which interface directly to the Z80 data bus.

The CTC chip is used when a program requires that certain operations be performed at fixed time intervals or at pre-set frequencies. In general, the relationship between the CTC and CPU can be regarded as that between a person and his or her watch. The CTC is a 28-pin chip with eight pins (D0 through D7) used as CPU data bus, seven pins used as CTC control, three pins as interrupt control, and another seven pins as channel signals. The remaining three pins are pin 24 (to which a 5-volt power is supplied), pin 5 (ground), and pin 15 (which receives a one-phase 5-volt clock pulse).

## 4 · 16 Power Supply

A power adaptor is supplied together with the MPF-I so that you can convert the higher voltage typically supplied by a wall outlet to 9V at 600mA.

The MPF-I requires a single 5V power supply at 500mA. A regulator is installed right beneath the socket for the power adaptor to convert 9-volts to 5-volts. A heat sink may or may not be attached to the voltage regulator to dissipate the heat of the voltage regulator. Don't touch the voltage regulator. It makes your finger uncomfortable.

## Questions

4-1. Turn to section 1.3 Physical Configuration of MPF-I User's Manual. Using the information on this page draw a circle around the Z8Ø CPU.

Physical Configuration

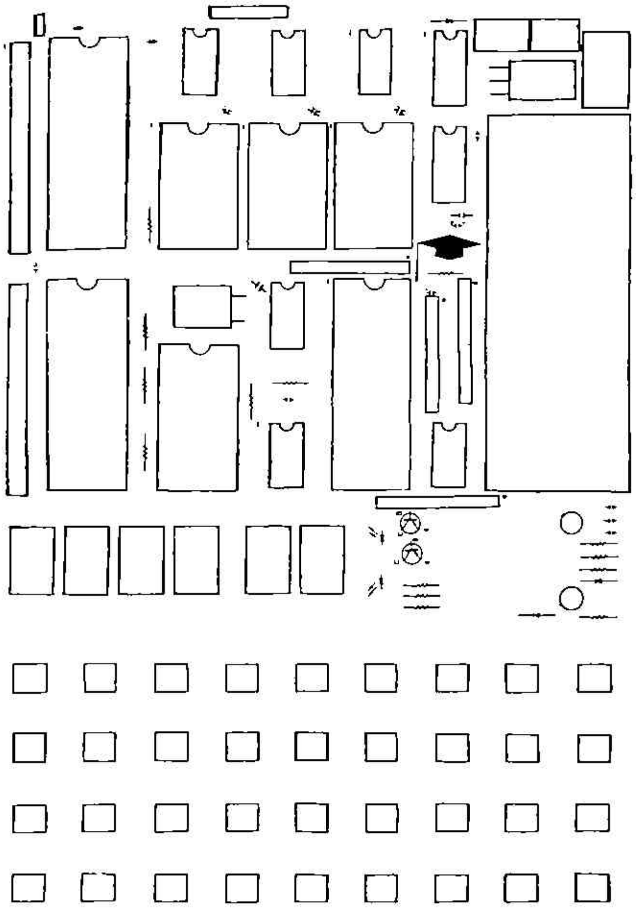


Fig 4-4

4-2. In Appendix B there are four pages of schematics. Look in the lower right hand corner.

| MULTITECH | | |
|---|---|---|
| TITLE: MPF-I | | |
| SHEET 2 OF 4 | DATE | REVISION |
| DRAWING NO. | 81/09/22 | A |

Fig 4-5

Below the title MPF-I there is an entry indicating which sheet you are reading. In the figure above this is sheet 2 of 4. Find sheet 1 of 4. Notice that to locate any component there are coordinates on the boarders Fig 4-6.

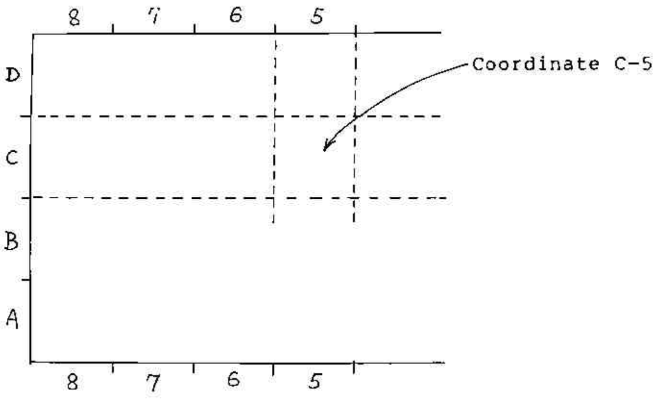



Fig 4-6

Locate the component at C-5. What is this component?____ This part also has a U number what is it?__________

4-3. The Z80-CPU transfers data in and out through its' data pins There are eight data pins that are all accessed at one time. The eight pins are grouped under the name data bus. Turn to the diagram CPU PIN-OUTS Appendix C page C-1. Locate the DATA BUS. D0 is the least significant binary digit and D7 is most significant binary digit. Fill in chart below

| BINARY DIGIT | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| PIN NUMBER | | | | | | | | |

When you filled in the chart above, you probably observed that the pin numbers for the data bus are not sequential. The pin numbers jump all around. There is no requirement that pin numbers for a bus be sequential.

4-4. Find the RAM in one of the sheets of the schematics in Appendix B (it is labeled U8). What sheet is the RAM on__. What are the coordinates of the RAM?____________. Around the edges of the chip are the pin numbers and their functions. In the center you will see HM6116. A 6116 is a type of RAM. Also on the chip is a memory address. The unit as delivered has the 6116 RAM located at addresses 1800H to 1FFFH.

4-5. Again refer to the MPF-I schematics. Find U6 the monitor ROM. What sheet is it on?__. What are the coordinates?__. Notice the type of allowable chips written on U6-- a 2516 or 2532. The 2516 option allows 2048 bytes or characters (2K=16K of bits) of information to be retained by the 2516. How many bytes would you think the 2532 chip allows to be retained?__________.

4-6. The Z80 CPU is able to address memory chips by connecting the address bus to the Z80 CPU and to the memory chip. The individual lines of the address are labeled A0 to A15. Find the address bus from the Z80 CPU (U1) to the monitor ROM at U6. Enter the pin connections of Z80 CPU and U6 in the chart below.

| ADDRESS BUS PIN NAME | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Z80 CPU (U1) PIN NAME | | | | | | | | | | | | | | | | |
| RAM (U6) PIN NAME | | | | | | | | | | | | | | | | |

Although it may be clear to you from reading the schematic the address (and data) lines travel under U6. This means that AØ of the Z8Ø CPU is connected to AØ of U7 and AØ of U8. Enter the corresponding pin connections in the chart below.

| ADDRESS BUS PIN NAME | A15 | A14 | A13 | A12 | A11 | A1Ø | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | AØ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Z8Ø CPU (U1) PIN NAME | | | | | | | | | | | | | | | | |
| RAM (U8) PIN NAME | | | | | | | | | | | | | | | | |

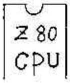
4-7. The data bus connects to several ICS just as the address bus does. Find the data bus on sheet 1 of 4. Enter the corresponding connections (pin numbers) in the chart below

| DATA BUS PIN NAME | D7 | D6 | D5 | D4 | D3 | D2 | D1 | DØ |
|---|---|---|---|---|---|---|---|---|
| Z8ØCPU (U1) PIN NAME | | | | | | | | 14 |
| ROM (U8) PIN NAME | | | | | | | | 9 |

The entire data bus is also used to access information from devices such as the keyboard. The 8255(U14) controls the keyboard so the data bus must be connected to this chip. This is so that the 8255 can send keyboard information to the CPU. Look at sheet 2 of 4 coordicates C-8 and D-8. You will see lines (wires) with the labels DØ to D7. Where did these lines come from? To the left of DØ through D7 is a parenthesis labeled SH1,3. SH stands for sheet. The data lines leave sheet 2 of 4 and connect to sheets 1 and 3. Can you find the connection on sheet 1? What are the coordinates? What are coordinates for the data bus on sheet 3 of 4?

## Answers

**4-1**




**4-2** At C-5 the Z8Ø-CPU. The U number is 1.

**4-3**

| BINARY DIGIT | D7 | D6 | D5 | D4 | D3 | D2 | D1 | DØ |
|---|---|---|---|---|---|---|---|---|
| PIN NUMBER | 13 | 1Ø | 9 | 7 | 8 | 12 | 15 | 14 |

**4-4** The RAM is on sheet 1 of 4.
The coordinates of the RAM are C-2.

**4-5** The ROM is on sheet 1 of 4.
The coordinates of the ROM are C-4.
The ROM can store (retain) 4Ø96 bytes.
(4K=32K bits).

**4-6**

| ADDRESS BUS PIN NAME | A15 | A14 | A13 | A12 | A11 | A1Ø | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | AØ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Z8Ø CPU (U1) | 5 | 4 | 3 | 2 | 1 | 4Ø | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 3Ø |
| RAM (U6) PIN NAME | NOT USED | | | | 18 | 19 | 22 | 23 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| ADDRESS BUS PIN NAME | A1Ø | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | AØ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Z8Ø CPU (U1) PIN NAME | 4Ø | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 3Ø |
| RAM (U8) PIN NAME | 19 | 29 | 23 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**4-7**

| DATA BUS PIN NAME | D7 | D6 | D5 | D4 | D3 | D2 | D1 | DØ |
|---|---|---|---|---|---|---|---|---|
| Z8ØCPU (U1) PIN NAME | 13 | 1Ø | 9 | 7 | 8 | 12 | 15 | 14 |
| ROM (U6) PIN NAME | 17 | 16 | 15 | 14 | 13 | 11 | 1Ø | 9 |
| RAM (U8) PIN NAME | 17 | 16 | 15 | 14 | 13 | 11 | 1Ø | 9 |

The coordinates of the data bus on sheet 1 of 4 are D-1.
The coordinates of the data bus on sheet 3 of 4 are C-7 and D-7.

# CHAPTER 5

## Introduction to Programming the MPF-I

## 5 • 1 Learn by Doing

You will now be guided through a series of examples from the MPF-I User's Manual. You should first key in the example and execute the program. But if you want to learn programming, you must do more. Each example will be analyzed--some examples in great detail. Whenever a new instruction occurs, you will be shown:

1) how to test if it is in the Z80 instruction set.
2) the correspondence between assembly code and object code.
3) what registers, flags and memory locations are affected by the instruction.
4) and finally the reason for using the instruction.

## 5 • 2 Flashing a Message

Turn to EXAMPLE 2 in section 5.10. Key in and execute this example. Does the program flash HELP US for 500 ms (1/2 second) and then go blank for 500 ms? Actually you should see HELP US for a longer time than 500 ms and blank screen for less than 500 ms. The program lights the screen for 500 ms but the display takes a period of time to extinguish (fade out) when they are no longer selected.

Hex

37 = H

## 5 • 3 Program Analysis

Exercise 5-1

Statement 1: flash 'HELP US'

You must understand you are writing your program in a highly readable form. Some words in your program will not be translated into an object program. An example is the comment statment, like statement 1. When using an assembler to translate your source program into object code, the comment statement must start with a semicolon. The semicolon signals the assembler to ignore the comment statement. Why use a comment statement? Comments are used to make the program understandable to readers and to programmers. Such statements are called documentation. A comment statement helps document a program.

Ex 5-2

CBE 93

3F = A

A7 = b

BE = 9

8D = C

B3 = d

BA 3

8F = E

ØF = F

1F = P

37 = H

85 = L

B1 = J

30 = I

02 = -

AE = 5

B5 = U

Statement 2: ORG 1800H

The ORG statement informs the assembler where to place the translated code. ORG stands for origin -- a beginning. When the assembler sees an ORG statement, it sets a counter which determines the location of each translated instruction. This location counter is advanced as each instruction is converted into object code.

Statement 3: LD HL,BLANK

This statement loads the address of BLANK into the register pair HL. To determine the address of BLANK, refer to line 19. BLANK is a label and thus is in the column (field) where labels are located. The address of LABEL, 1826, is given by the lefthand column. The location counter is responsible for calculating the values in this column. It has now been determined that statement 3 loads the value 1826 into the HL register pair. The H can be assumed to stand for high, thus the high byte, 18, is loaded into the H register. L means low, so the low byte, 26, is loaded into the L register.

When you are writing a program, you need to know what the instruction set is. Can the register pair HL be loaded with a value given in the instruction (BLANK)? This value is called an immediate, because you can look at the object code and immediately see the numbers being loaded into the registers.

To determine the legality of LD HL,BLANK, you need to know two facts: 1) is there an H and an L register which can they be paired and 2) is the instruction allowable. To determine the first fact, turn to Appendix C and find the page titled Z-80 CPU REGISTER CONFIGURATION. Yes, near the top of the page under MAIN REGISTER SET you see H and L. The Z80 REGISTER CONFIGURATION is also shown in fig 5-1 (and fig 5-2). Now look in Appendix C for the page with the title 16-BIT LOAD GROUP 'LD' 'PUSH' and 'POP'. Find SOURCE at the top of the chart then REGISTER below SOURCE. Under REGISTER the fourth entry over from the left contains HL, thus H and L may be paired. But are H and L being used as a source in the instruction LD HL,BLANK? No, the BLANK is being loaded into HL, therefore, HL is a destination. Looking on the left side of the chart, find DESTINATION then REGISTER. The fourth entry from the top (under REGISTER) is HL. So HL can be used as a destination. Can an immediate value be loaded into HL? Travel from left to right in the row labeled HL until you come to the column labled IMM.EXT (immediate extended). At the intersection of the row and column, there is value (21). A box with a value in it means that the instruction is allowed. Each "n" in the box stands for one byte. The upper "n" is the value to be loaded into L, and the lower byte is the value to be loaded into H.

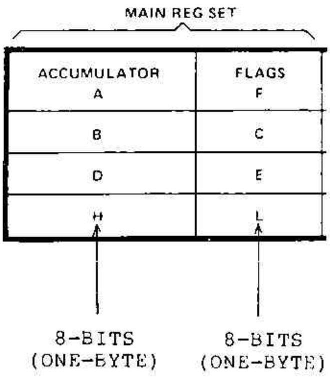

MAIN REG SET
ACCUMULATOR
A
FLAGS
F
B
C
D
E
H
L
8-BITS
(ONE-BYTE)
8-BITS
(ONE-BYTE)

Fig 5-1

## 16-Bit Load Group

(4)

| | | | SOURCE: REGISTER | | | | | | | IMM. EXT. | EXT. ADDR. | REG. INDIR. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | AF | BC | DE | HL | SP | IX | IY | nn | (nn) | (SP) |
| DESTINATION | REGISTER | AF | | | | | | | | | | F1 |
| | | BC | | | | | | | | 01 n n | ED 4B n n | C1 |
| | | DE | | | | | | | | 11 n n | ED 5B n n | D1 |
| | | HL | | | | | | | | 21 n n | 2A n n | E1 |
| | | SP | | | | F9 | | DD F9 | FD F9 | 31 n n | ED 7B n n | |
| | | IX | | | | | | | | DD 21 n n | DD 2A n n | DD E1 |
| | | IY | | | | | | | | FD 21 n n | FD 2A n n | FD E1 |
| | EXTERNAL ADDRESS | (nn) | | ED 43 n n | ED 53 n n | 22 n n | ED 73 n n | DD 22 n n | FD 22 n n | | | |
| PUSH INSTRUCTIONS | REGISTER IND. | (SP) | F5 | C5 | D5 | E5 | | DD E5 | FD E5 | | | |

(1) (2) (3) (5)

NOTE: The Push & Pop Instructions adjust the SP after every execution.

Fig 5-2

(1) (3)

16-Bit Load Group

| Mnemonic | Symbolic Operation | S | Z | | H | | P/V | N | C | Opcode 76 543 210 | Hex | No. of Bytes | No. of M Cycles | No. of T States | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LD dd, nn | dd ← nn | • | • | X | • | X | • | • | • | 00 dd0 001 ← n → ← n → | | 3 | 3 | 10 | dd Pair: 00 BC, 01 DE, 10 HL, 11 SP |
| LD IX, nn | IX ← nn | • | • | X | • | X | • | • | • | 11 011 101 00 100 001 | DD 21 | 4 | 4 | 14 | |

(2) (4)

Fig 5-3

The correct form for the source code can be found on the next page titled 16-BIT LOAD GROUP (see fig 5-3 also). On the leftmost column is the mnemonic column. Mnemonic means assisting or intended to assist the memory. below the title MNENOMIC is the form for load immediate, LD dd, nn. The LD, of course, means load. "nn" is the immediate value - BLANK (1826) in statement 3. To understand "dd" locate the column labeled COMMENTS on the far right. "dd" tells the programmer what register pairs can be used in the 16 bit load immediate instruction. Thus;

```
LD BC,nn
LD DE,nn
LD HL,nn
LD SP,nn
```

are allowed. To complete the LD HL,nn instruction, simply fill the value for nn , e.g., LD HL,BLANK. LD HL,1826H would produce the same result.

If you are hand translating the assembly language instructions you must use the chart on the previous page. Remember that 21nn that was found at the intersection of HL and IMM.EXT 21 is called the opcode (operation code). The translation gives

```
2 1 2 6 1 8
```

Why wasn't the result of the translation

```
2 1 1 8 2 6
```

Because the low byte 26 must follow the opcode, then the high byte 18. Don't fight it! You must write values this way in Z80 coding. LD HL,BLANK translates into a 3 byte instruction. The location counter will be advanced by 3 in preparation for the next instruction 1800 + 3 --> 1803. In summary:

| Location Counter | Object Code | Statement No. | Source Code |
|---|---|---|---|
| 1800 | 212618 | 3 | LD HL, BLANK |

Ex 5-3

Statement 4: PUSH HL

The PUSH instruction is used to move the contents of a register pair or a 16-bit register to a specific place in memory. To determine the assembly language code mnemonic, turn to Appendix C and proceed to the chart 16-BIT LOAD GROUP. Travel down the leftmost column labled Mnemonic until the mnemonic PUSH is located. Since "qq" means that BC, DE, HL and AF are allowed, this is the correct form. To translate the instruction into machine language, refer to the chart 16-BIT LOAD GROUP 'LD' 'PUSH' and 'POP'. The source is the content of the HL register pair. Find SOURCE, Register and then HL. For destination find the title PUSH INSTRUCTIONS at lower left hand part of the page. Where the column HL and row PUSH INSTRUCTIONS meet is the value E5. This is the value you will enter. This one byte instruction advances the location counter by one 1803+1 --> 1804.

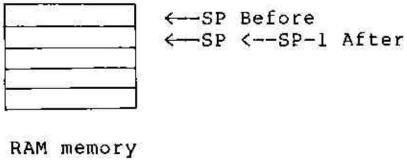
Details of the push instruction.

A PUSH instruction transfers the contents of registers to a region in memory called the stack. The stack is defined by a pointer called a Stack Pointer (sp). In EXAMPLE 2 the stack pointer was set by the monitor before you began execution of the program.

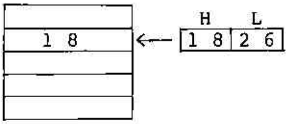
STEPS IN THE EXECUTION

OF PUSH HL

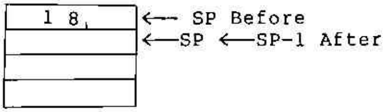
STEP 1: DECREMENT THE STACK POINTER



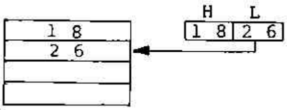

STEP 2: PUSH H ONTO THE STACK




STEP 3: DECREMENT THE STACK POINTER




STEP 4: PUSH L ONTO THE STACK




Ex 5-4

Statement 5: LD IX, HELP

This statement is very similar to statement 3. It is a load immediate instruction. The 16 bit register IX is being loaded instead of the register pair HL. The immediate value is 1820H (see statement 13). There is something new besides using index register IX as the destination. This instruction has two opcodes. Find the object code for the instruction by turning to the 16-BIT LOAD GROUP 'LD' 'PUSH' and 'POP' in Appendix C. The intersection of the source IMM.EXT and destination IX shows DD21nn. The two opcodes are DD and 21. The reason for the double or extended opcode is because the Z80 CPU, designed by Zilog, is an improved 8080 (an earlier CPU designed by INTEL). Zilog wanted the Z80 CPU to be able to execute all of the 8080 instructions plus the ability to execute new instructions. Some opcodes were not used by the 8080 CPU. If only one opcode was used in the empty slots (unused 8080 opcodes), only a few new instructions could be added. A double opcode allows the DD to be followed by one of 256 different codes (00H to FFH). Now in place of one unused opcode, many new instructions can be added. If you look at the row labeled IX, you will see that all the instructions have as the first opcode a DD. HELP is a label in statement 13. The value of the location counter at this point is 1820. When you translate LD IX,HELP to object code, the nn (2 bytes) will contain 1820. The object code for LD IX,HELP is DD 21 20 18. Don't forget the lower order byte 20 is written first followed by the high part of the address 18. LD IX,HELP is a four byte instruction. The location counter will advance by 4 1804 + 4 = 1808

Ex 5-5

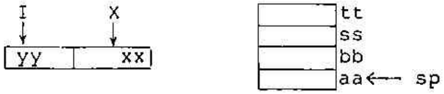
Statement 6: LOOP EX (SP),IX

The instruction asks the computer to EXchange the two byte pair currently pointed to by the stack pointer with the contents of the IX register.

BEFORE:

(RAM memory)

AFTER:




(RAM memory)

The first time this instruction is executed, the stack will contain 1820H and IX will contain 1826H. Because of the exchange, the next time this instruction is used the stack will contain 1826H and IX will contain 1820H. The action of EX (SP),IX is to make index register IX alternate between pointing to the message HELP US at 1820H and the blank display at 1826H. Enclosing an instruction in parentheses indicates a memory reference. The stack pointer is enclosed by parentheses (SP) thus the stack points to memory.

Ex 5-6

Statement 7: LD B,50

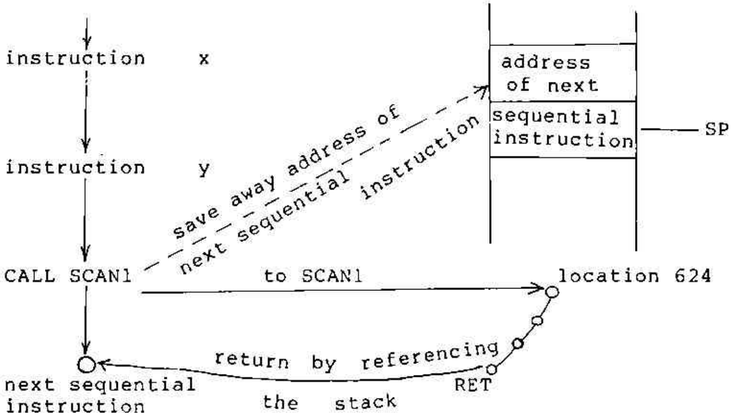
The constant (immediate value) is loaded into the B register. The exercise Ex 5-7 will explain this instruction.

EX 5-7

Statement 8: CALL SCAN1

A series of instructions which perform a definite task is called a routine. A program consists of one or more routines. The monitor contains several routines which the user may wish to access. SCAN1 is a monitor routine which will (as one of its actions) display the area pointed to by IX. The display consists of 6 sections so IX will point to a six byte region. A routine accessed by another routine or program can be called a subroutine. The CALL instruction is the preferred method to access a subroutine.

The CALL instruction breaks the sequential processing of instructions by transferring control to a new address. In statement 8 the new address is the entry point into the routine SCAN1. The execution of SCAN1 is terminated by a return (RET) instruction. The return instruction is used to order program control to continue just after the CALL instruction.




How a CALL-RET Works CALL SCAN1

In reality when CALL SCAN1 is executed, the contents of the program counter (PC) which already points to the next sequential instruction are saved on the stack. The contents of the PC (180F), in Example 2, are pushed (saved) on the stack. Now the program counter is loaded with the subroutine address given the CALL SCAN1 instruction. (in this example the address is Ø624H, SCAN1). Program control is now transferred to SCAN1. When the return (RET) instruction in SCAN1 is executed, the program counter will be loaded from the stack. The value on the stack is the address of the next instruction after SCAN1, so control returns to location 18ØFH.

After the above explanation you may have forgotten what's happening. The call to SCAN1 will use the six bytes at BLANK to control the screen (displays). Zeros are sent to the display in the MPF-I, which turns off all the segments in a display. So BLANK blanks the screen, but only for a short time.

EX 5-8

Statement 9: DJNZ HELFSEG

Statement 9 provides the solution to the very short time that SCAN1 will blank out the screen. What is needed is a method of repeating statement 8 which will again display the current pattern that the IX register is pointing to.

The DJNZ instruction will:

1) Decrement the B register. B was loaded with a 50 (decimal) so it will now contain 49(decimal).

2) Compare B with zero.

3) If B is not equal to zero, program control is transferred to the location given in the operand field. The operand field contains HELFSEG so,the program continues at the statement containing HELFSEG as a label.

From the above you can see that statement 8 will be executed 50 times until B becomes zero. When B does equal zero, execution continues sequentially at statement 10. Executing statements 8 and 9 fifty times will hold a pattern of the screen for about 500 ms.

Ex 5-9

Statement 10: JR LOOP

The J in this statement means Jump. A jump is a transfer of control. The R means jump relative from where the program counter is at this time. The program counter has advanced to location 1813. The operand Loop indicates a relative jump to the statement with the label LOOP--statement 6.

Ex 5-10

Statement 11:

This is a sneaky way to get a line with nothing but a semicolon. This comment line without a comment makes the program easier to read.

Statement 12: ORG 1820H

Reset the location counter to 1820H. The following data will be located at hex location 1820 and up.

Statement 13 to 24

DEFB means define a byte. That is: reserve a location and enter a particular pattern at this location. The DEFB's are used to generate display patterns (characters).

Ex 5-11

Statement 26: SCAN1 EQU 0624H

This statment is used to inform the assembler whenever you see the operand SCAN1 put the hexadecimal number 0624 in its place. EQU means equate.

Statement 27: END

An end statement informs the assembler that there is nothing left to translate into object code.

It is possible to know what every statement in a program does and not understand what the program is doing. Lets trace the major actions of EXAMPLE 2.

The first time statements 1 to 6 are executed IX will point to BLANK and a pointer to HELP is on the stack. Statements 7 to 9 will keep the screen blank out for about 500 ms. Then statement 10 transfers the program control to statement 6. Statement 6 will make IX point to HELP and put a pointer to BLANK on the stack. Statements 7 to 9 will display HELP US for about 500 ms. Again statement 10 transfers control to statement 6. An exchange of the contents of IX and the stack occurs so that now blanks will be displayed for about 500 ms. You must press either RS or MONI to stop the alternating display.

Ex 5-12

TERMINATING A MESSAGE

Turn to EXAMPLE 1 in section 5.1Ø. Key in and execute the example. In each EXAMPLE only new features will be discussed. There are three new features in this example. One, only one screen pattern is displayed. In Example 2, HELPUS alternated with a blank screen. Two, a different routine, SCAN is used to display the message. Lastly, the program can be stopped by pressing a key, namely the STEP key.

Program analysis

Statement 3: LD IX,HELP

Only one message is displayed and no blanking will occur, thus IX is loaded with a pointer (an address) to the message. When either SCAN or SCAN1 are called the 6 byte group pointed to by IX will be displayed.

Statement 4: CALL SCAN

You should read the explanation of SCAN in section 5.3. You will discover:

1) IX points to the display buffer.

2) The message (contents of the display buffer) will be displayed until a key is pressed.

3) The A register will contain the internal code of the key pressed. See Statement 5 below for a discussion of key codes.

4) The address of SCAN in the monitor is Ø5FEH.

Statement 5: CP 13H

How can the continuous display be terminated? Decide on one key to terminate the program. In this program the STEP key has already been choosen. The monitor program in con-

junction with hardware is designed to return a unique internal code for any key (except RS, MONI, INTR, and USER) pressed. Actually a code dependent upon the position of key is returned first. The position code is converted into an internal code when using SCAN. To determine the internal code for STEP - or any other key - refer to Appendix A section 2; Internal code (CALL SCAN): You will find STEP in the second row from the bottom and the fourth column from the right. The code is 13H.

EX 5-13

What is needed is a method of testing the A register for a particular code (key value). The compare instruction - (CP 13H) compares the value 13H with the contents of the A register. The details are:

1) Put a copy of the A register into a temporary register.

2) Subtract 13H, or any value given as an operand, from the copy.

3) If the copy of A equals the test (choosen) value set the zero flag. If the copy of A is less than the chosen value, set the sign flag. Thus testing a maximum of two flags can determine how the A register compares to a particular value-when the compare instruction is used before testing.

In summary:

A = test value; zero flag is set.

A < test value; sign flag is set.

A > test vaue; neither the zero or sign flag is set.

Actually, using the results of the compare instruction is easier than thinking all about flags as you will see in the description of statement 6.

EX 5-14

The compare instruction does affect flags. Turn to the second page of the 8-BIT ARITHMETIC AND LOGICAL GROUP. Find the set of columns labeled Flags. Now find the row labeled CP s. We will analyze the meaning of the first two columns under flags. The S cloumn means sign (of the comparison). There is an up down arrow at the CP s position in this column.

Up arrow means if the result was negative, then the flag will be set. In plain terms when the A register is smaller than the test value, the sign flag is set. The down arrow indicates the result was either zero or positive and the flag will be cleared. Again in plain English, the value of the A register was not less than the test value. Remember set means 1 and reset eans zero. The Z column means zero. If A equals the test value, the flag will be set (up arrow). If A is not equal to zero, the flag is reset (down arrow).

Statement 6: JR NZ,DISP

The program should be designed to repeat the current display unless any key but STEP is pressed. The compare statement CP 13H resets the zero flag if any key but STEP is pressed. Then all that is needed is an instruction that will jump back to CALL SCAN, labeled DISP, when the zero flag is not set. JR NZ,DISP says transfer program control to DISP if the result of the test (or any operation that affects the zero flag) was non-zero (NZ). If the STEP key was pressed, then statement 6 does not break the sequential flow of instructions and the next instruction executed is HALT. When a program cycles again and again through the same sequence of instructions it is said to be looping. When a test does not break the sequential execution of instructions, the slang expression 'fallen thru' (to the next instruction) is used. In this example, you could have avoided understanding flags. Understanding the interaction of

CP 13H and JR NZ,DISP

would be sufficent. Do a compare. If the A register equals the operand of the CP instruction, then a JR NZ, label will not jump to label. If the A register dosen't equal the operand, then JR NZ, label will transfer control to the instruction with the label.

Ex ~ 15

Statement 7 HALT

The computer has stopped looking for commands to execute. The screen will go blank. To regain control you must press either MONI or RS.

Ex 5-16

Using (Calling) Two Monitor Routines

Turn to EXAMPLE 3 in section 5.10. Key in and execute the example. Read the instructions given below the listing.

Statement 4 LOOP CALL SCAN

EX 5-17

Statement 5 LD HL,OUTBF

EX 5-18

Statement 6 CALL HEX7SG

HEX7SG is a routine residing in the monitor. Turn to section 5.5 and read about HEX7SG. The sequence of the actions for a particular key press will now be described. Assume that you pressed the RELA key. Statement 4 CALL SCAN will put the internal code for RELA into the A register.

1D ——→ A register

HEX7SG first converts the D into a 7-segment display format

D B3=

(D converts to B3) and then stores the byte B3 at location OUTBF. Effectively, statement 10 now reads

```
OUTBF DEFB B3H
```

Next HEX7SG converts the 1 into a seven segment display format and stores the result at OUTBF+1

1 30 =

EX 5-19

statement 10 and 11 now read

```
OUTBF DEFB B3H
      DEFB 30H
```

Statement 7 JR LOOP

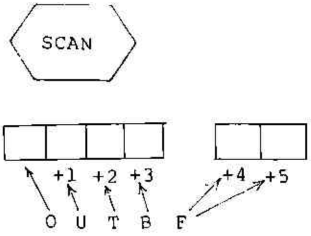
The jump relative command will jump to location LOOP. Statement 4 (again) CALL SCAN

Summary

Remember that SCAN will output the contents of the display buffer and cycle until a key is pressed. When a key is pressed the internal code of the into key is loaded into the A register. What is in the display buffer. The first two bytes contain the display codes for the bytes in the A register. HEX7SG converted contents of the A register into display code.

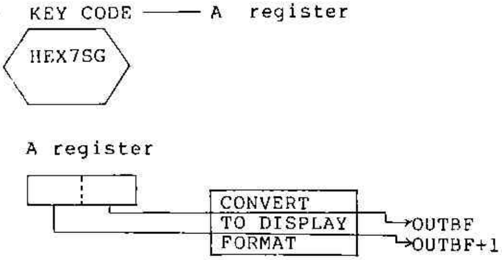
1. Actions of SCAN

1.



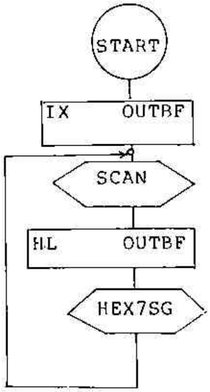

2. Actions of HEX7SG

2.




Open the MPF-I Experiment Manual (Software/Hardware) to Introduction to Designing Microcomputer Programs. Read B. Flowchart. One additional symbol you should know is

A flowchart of EXAMPLE 3 is




LD IX with the address of OUTBF

Execute the routine SCAN

Load IY with the address of OUTBF

Execute the routine HEX7SG

LOOP

EX 5-20

A DISPLAY CONVERTER

Turn to EXAMPLE 4 in section 5.10. Key in and execute this example.

EX 5-21

POLICE SIREN

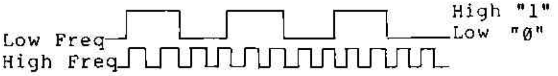
Turn to EXAMPLE 5 in section 5.10. Key in and execute this example.

EXAMPLE 5: Simulate a police car siren

The siren produced by this program consists of two tones, each one lasting 0.73 sec. The two frequencies are 256Hz and 352Hz.

Statement 3 LOOP LOOP C,Ø

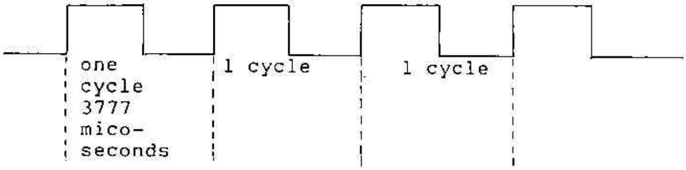
The frequency is controlled by the value in C. The larger the value of C, the lower the frequency. The note produced is a square wave. The wider the square wave, the lower the note.




The square wave is held high for the number of counts in C, and then low for the same count. But the test for the time to hold C high or low is done after subtracting one from the value of C. What is one less than C? For all values except zero, it is simple, e.g., 192-1 = 191 FEH -1 = FDH. What is one less than zero? When the computer is using plus and minus numbers, FFH equals -1. Thus one less than zero is FFH. But the test for the square wave generator doesn't use signs, therefore, FFH is equal to 255 decimal. In statement 3, the C register is loaded with zero. This will generate the biggest number when tested by the tone routine and the lowest possible frequency using the monitor tone routine. The calculation given below the code in EXAMPLE 5 shows this frequency to be 265Hz ( Hertz=cycles/sec). This is close to the middle of a piano keyboard (middle C). Occasionally, computer programs use a trick, like one less than zero having the effect of being a large number.

Statement 4: LD HL,ØCØH

How long will the tone at 256Hz sound? Another calculation reveales the period of one cycle at 256Hz to be 3777 micro seconds.



The value in HL, when used by TONE, determines the number of cycles and thus the length of the sound at a particular frequency. At 256Hz a value of 2 in HL produces a length of 7554 micro seconds--less than a hundred of a second. In this example, HL contains the value ØCØH which equals 192 decimal. The length of the sound is 3777 micro sec. x 192 = Ø.73 sec.

Statement 5: CALL TONE

In specifying parameters (values) for the TONE, you already know that the frequency is set by the value in C and the length of a sound is contained in HL. Reinforce your knowledge of TONE by reading section 5.7. Do not avoid studying how to calculate the frequency and the tone length.

EX 5-22

## Memory Checking

Turn to Memory Check section 6.1. Key in and execute the program. Note the display and the condition of the HALT LED. The HALT LED is a red light to the right of the displays. Why did you run this program? Read further.

What are the areas of employment in the microprocessor field? A partial list could be:

1) Chip (integrated circuit) designers -- the Z8Ø CPU is an example of a chip requiring a high level of technology.

2) Hardware designer - the people who determine how the components will interface.

3) Software programmers -the MPF-I the monitor is a software program held in a PROM.

4) Applications programmers - The Music Box program (Experiment - 18) is an application program.

Some additions to the list would be a sales staff. But something very important (and a growing field) is missing. The people who design tests. The various ICs and the computer as a whole should be tested. Testing starts with the components. Your Z8Ø CPU is tested at the factory. The tests guarantee that the Z8Ø CPU will function over a specified voltage and temperature range. Two built-in tests are provided for your convenience - a PROM test and a RAM test.

## EPROM Testing

The information in a PROM doesn't disappear when the voltage is removed. Some PROMs, EPROMs can be erased by applying ultraviolet rays. PROM tests take advantage of the fact that information in a PROM doesn't change easily. Imagine a very small PROM containing only 4 location (bytes). Assume that the bytes are Ø2,Ø1,Ø3 and ØØ. Adding up the bytes would give a sum of Ø6. If a byte contains an incorrect value, the sum would be different. For example, if the last byte were Ø1 instead of ØØ, then the sum would be Ø7. Since the sum is being used to check the PROM, it is called a checksum. Even with only four bytes, the sum might be larger than the largest value that a byte can contain. Any carries out of the byte are ignored. In spite of

throwing away the carries,the sum in the byte will always be the same in a healthy PROM and circuit. If you had a PROM with 2048 decimal locations of which 2047 are needed, you have a spare byte. Could the spare byte be **used** to produce a useful checksum? Yes, by adding the correct value to the checksum of 2047, the result can be made to equal zero. As an example, adding 2 to a hexadecimal result FEH produces a carry (which is ignored) and a byte containing a zero. If the PROM routine changed, the test programmer changes the extra byte to guarantee a result of zero. Now the PROM test only has to be written once. It is always enough to add up all the bytes in the PROM and test for a zero result. Your Micro-professor PROM test routine uses this add-up-to-zero method.

Turn to the EPROM test section 6.1

Initialization

The statements

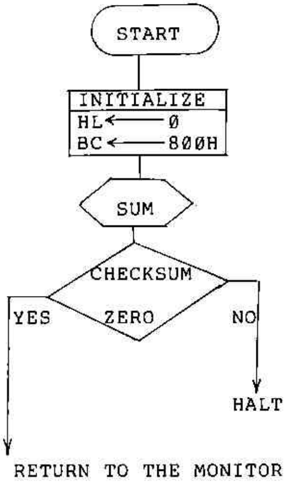
```
LD HL,0
LD BC,800H
```

are called initialization code. The HL pointer is set to the beginning address of the EPROM--zero in MPF-I. The BC register pair is set to the number of bytes to be tested. **The MPF-I monitor PROM holds 2K bytes, which equals 800** hexadecimal locations. The, CALL SUM, is made to a subroutine which adds up all the bytes in the monitor EPROM. When the subroutine SUM completes execution a RET instruction is executed and control is returned to the relative jump statement

```
JR  Z,SUMOK
```

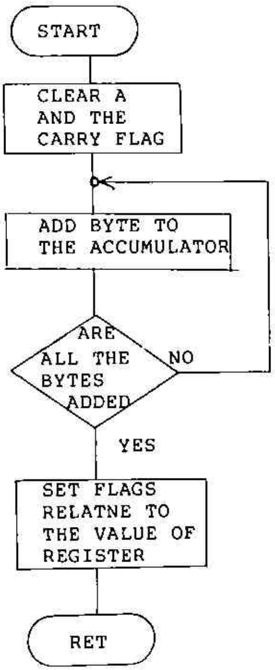
If the result of summing all the numbers in the A register is zero, the relative jump on zero will transfer control to location SUMOK. The command at SUMOK will transfer control to beginning of the monitor location zero. If the sum of the bytes in the PROM was not zero, then the jump relative command will not transfer control and program execution continues at the next command which will halt the processor (MPF-I).

ROM TEST Flowchart




| STEP | INSTRUCTIONS |
|---|---|
| 1 | LD H,Ø;BEGINNING OF PROGRAM<br>LD BC,8ØØH;PPOGRAM SIZE |
| 2 | CALL SUM |
| 3 | |
| 4 | HALT |
| 5 | RET |

SUBROUTINE SUM FLOWCHART




| STEP | INSTRUCTIONS |
|---|---|
| 1 | XOR A |
| 2 | ADD A,(HL) |
| 3 | CPI<br>JP PE,SUMCAL |
| 4 | OR A |
| 5 | RET |

EX 5-23

The subroutine SUM

The flowchart of SUM shows the actions performed by SUM. Read the flowchart then proceed to the detailed explanation of each command given below.

XOR A

XOR means exclusive OR. An exclusive OR operates on two bytes. The contents of the A register is always one of the bytes, the other byte is given in the operand field. The command XOR B will exclusively OR registers A and B. When bytes are exclusively ORed together, 8 bit pairs are ored to form a one-byte answer.

Assume A contains 10101100 and B contains 11001010 then

XOR B

gives

```
8 bit pairs  1 1 0 0 1 0 1 0      register B
             1 0 1 0 1 1 0 0      register A
            -----------------
             0 1 1 0 0 1 1 0      result
```

What do you observed whenever the bits in A and B were the same? The answer is zero. Whenever the bits were different, the answer is 1. The "truth" table below shows this relationship

| A | B | XOR B |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

XOR A means exclusively OR A against A. All the bits will be the same, thus the contents of A will be zero after XOR A. XOR A is a sneaky way of clearing A to zero.

EX 5-24

ADD A,(HL)

This instruction adds the contents of the location pointed to by HL to the accumulator register A. The first time the instruction is executed, HL points to the first byte of the monitor EPROM. The first byte of the EPROM is added to A, A <-- Ø + first byte.

## CPI

The compare and increment instruction will:

1. Compare the contents of the A register with the location pointed to by HL. This feature is not used by the subroutine SUM.

2. Increment the HL register pair.

3. Decrement the BC register pair.

4. Test BC for non-zero, after it has been decremented. If BC is non-zero, then set an indicator. The indicator is called a flag. The flag used is the P/V flag. The P/V flag is used in several ways. The next instruction will show you one use for P/V.

JP PE,SUMCAL

The JP PE,SUMCAL instruction orders the computer to transfer control to SUMCAL, if the parity flag is set. Set means that a "1" is in the flag.

| | P/V | Comments |
|---|---|---|
| Set | 1 | also called on |
| Reset | Ø | also called cleared |

The instruction CPI influences the actions of JP PE,SUMCAL. If BC is not zero, then program control transfers back to SUMCAL. BC was set to the number of bytes to be tested. The result of CPI and JR PE,SUMCAL working together is that control will be transferred to SUMCAL until all of the bytes have been summed up. CPI and ADD A,(HL) also work together each time CPI is executed. HL increases by one. If control is transferred to ADD A,(HL), the next byte is added to register A.

OR A

The results of executing an OR instruction can be determined by using the "truth" table below

| A | B | OR B | Operands |
|---|---|---|---|
| 1 | 1 | 1 | Both one |
| 1 | Ø | 1 | One Zero |
| Ø | 1 | 1 | One Zero |
| Ø | Ø | Ø | Both Zero |

The conclusion you should draw from the table is that unless both operands are zero, the answer is one. Another conclusion is that if both operands are the same, the result will be the same as the operands.

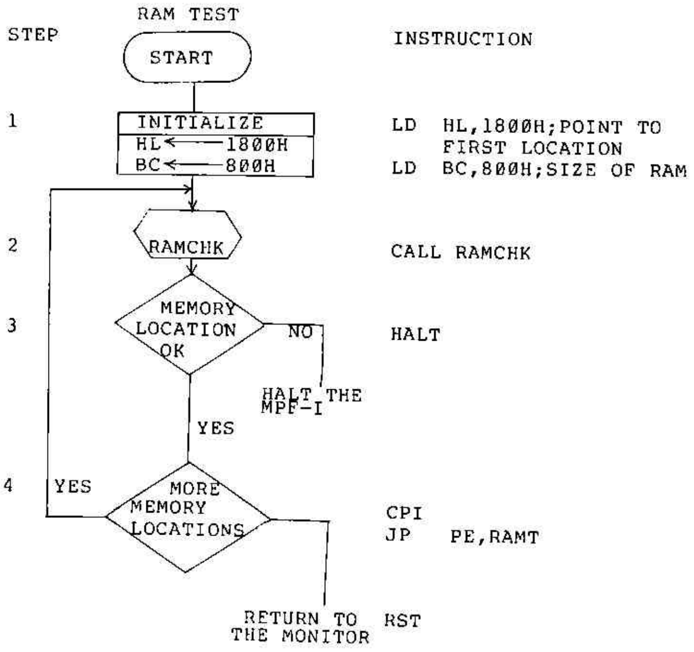
1 OR 1 gives 1          Ø OR Ø gives Ø.

ORing A against A will not change the value of A. For example

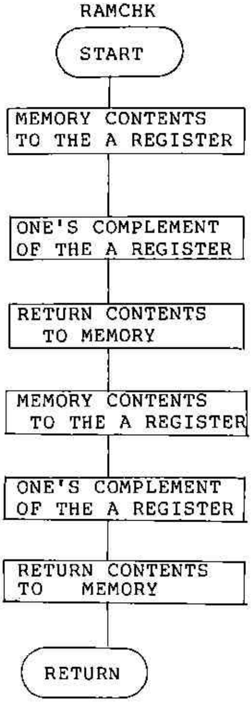
```
OR A   11ØØ1Ø1Ø       Register A

       11ØØ1Ø1Ø       Register A
      ----------
   A   11ØØ1Ø1Ø       Result in Register A
```

Then why OR A? Because another action occurs. Certain flags are set whenever an OR operation occurs. The flag settings depend upon the result of the OR operation. If the result of the OR operation is zero, then the zero flag is

set. This zero flag is what we are interested in. If the checksum was zero, then A contains zero. ORing zero against zero gives zero with the zero flag set. After the instruction RET is executed, the next instruction is JR Z,SUMOK instruction. This, of course, tests the zero flag. If it is set, control is transferred to SUMOK. Look again at your flowchart.



SUBROUTINE RAMCHK FLOWCHART




| STEP | | INSTRUCTIONS |
|---|---|---|
| | START | |
| 1 | MEMORY CONTENTS TO THE A REGISTER | LD A,(HL) |
| 2 | ONE'S COMPLEMENT OF THE A REGISTER | CPL |
| 3 | RETURN CONTENTS TO MEMORY | LD (HL),A |
| 4 | MEMORY CONTENTS TO THE A REGISTER | LD A,(HL) |
| 5 | ONE'S COMPLEMENT OF THE A REGISTER | CPL |
| 6 | RETURN CONTENTS TO MEMORY | LD (HL),A |
| | RETURN | RET |

EX 5-25

## RAM Testing

A RAM is designed to have its memory cont altered. This property is used when testing RAMs address the following procedure is followed. Lc from memory into the A register. Change each zer "1" and each one bit to a "Ø". This is cal... a one's complement. Put the complemented byte back into the original memory location. Load the complemented byte back into the A register. Again perform a one's complement and put the result back into the orginal memory location. Now compare the byte in the memory location with the byte in the A register. A failure indicates a bad memory, or possibly a bad address, bad data lines, or the CPU is not decoding instructions correctly .

Why does this program work? That is how can it test a RAM? A bad RAM chip has to exhibit a failure by returning an incorrect bit or bits when read. Assume in the frame shown below that bit 1 is stuck low (will not go high) and that bit 6 is stuck high (will not go low).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø |
|---|---|---|---|---|---|---|---|
| | H | | | | | L | |

Ram memory location

When the location is read into the A register and complemented, bit 1 will go high and bit 6 will go low.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø |
|---|---|---|---|---|---|---|---|
| | L | | | | | H | |

A register

Writing back the contents of A into the same memory location will not change bits 1 and 6 in memory. Comparing the contents of A with memory will give a non-zero result bits 1 and 6 at the of two locations are not equal.

If the one's complement instruction appears to detect errors then why are there two complement instruction? If you have a healthy RAM, after testing all the memory locations, it should be the same. With a single complement, they won't be complementively twice restores each healthy memory location to its origianl values.

EX 5-26

EX 5-27

## Questions

### 5-1

Change the comment statement to read we don't need any help. How would you separate the comment statement from the rest of the program?

### 5-2

How would you make a program start at 1900H ?
What would the statement ORG C000H do?
What is the effect of starting a program at 100H, ORG 100H ?

### 5-3

Use the 16-BIT LOAD GROUP charts in Appendix C to answer the following questions. What is the opcode for LD DE, BLANK ? How many bytes in the instruction LD BC,1826H? Is the instruction LD AF,BLANK allowed? Is the instruction LD DH, BLANK allowed? Is the instruction LD SP,BLANK allowed?

### 5-4

Show with a drawing all the steps in the PUSH BC instruction. Do the same for the PUSH IX instrucion. Hint : use Appendix C 16-BIT LOAD GROUP SP-1 <- IXH means that the high byte of the IX register (a 16-bit register) is put on the stack first. Can a constant be pushed upon the stack? Label the fields in the listing shown below

```
[    ]  [          ]  [     ]  [                    ]
1803      E3             4        PUSH   HL
```

### 5-5

How can you verify that LD IX,HELD is the correct form of the assembly language source statement? Using the table 16-bit LOAD GROUP Appendix C, find the column Symbolic Operation. What does the entry for LD IX, nn indicate?

**5-6**

Using Appendix C find the exchange instruction. What is the title of this group? What is the opcode for EX (SP),IX ? The DD again means? The location counter was 1808 what will it change to after the EX (SP),IX instruction?

**5-7**

The B register is an eight bit (one byte) register. How many bits are loaded in the LD B,50? What instruction group will give the object code for LD B, 50? Find the correct group in Appendix C.
What is the object code for LD B,50?
What is the source label ? What is the DESTINATION label ?
How many byte instruction is LD B,50 ?
Why does the 50 in LD B,50 translate to 32 in the object code ? The title of the immediate column in the 8-bit LOAD GROUP 'LD' is IMME., and the title in the 16-bit LOAD GROUP 'LD' 'PUSH' AND 'POP' is EXT.IMME. Why are the titles different ?

**5-8**

Find the CALL AND RETURN GROUP on the same page with the RESTART GROUP-Appendix C. You will CALL SCAN1 ununconditionally. The condition column is labeled UNCOND. The choice of the correct row should be easy, what is it? What is the opcode ? How many bytes is the instruction? What goes in nn? What is the object code for CALL SCAN1?

**5-9**

Read section 3.3.4 Relative Address Calculation. Try using the RELA on the DJNZ statement in this program. What is value of S ? What is value of D ? Find the JUMP GROUP in Appendix C. What is the opcode of DJNZ? THE e-2 means the relative distance to be jumped. In statement 9, the value of this byte is FB. Explain what this value means.

**5-10**

Find the JUMP GROUP in Appendix C. Now locate the row labeled JUMP 'JP' relative. Note the ' P' should read 'JR'. The first column under condition UNCOND is the correct column. What is the opcode? The second byte of the object code contains F5. How many bytes backward does this value represent? Show how to compute where the JR instruction jumps.

**5-11**

Study the display formats in Appendix A. Now change the screen display from HELPUS to all 8's. Display your initials. Use blanks in any position not occupied by your initials.

**5-12**

Projects

Some of the projects suggested in this paragraph may be beyond your abilities at this time. Instructions not yet explained may be needed. You may want to start designing your program now. Or experiment with altering instructions. How can you alternate alpha messages on the display? How would you put a blank message between alternating alpha messages? How can you have messages which are on the screen for different periods of time? Design a program which will move a display across the screen.

**5-13**

Give the internal codes for

| Key | Ø | 1 | 5 | Go | MOVE | MONI |
|---|---|---|---|---|---|---|
| Code | [ ] | [ ] | [ ] | [ ] | [ ] | [ ] |

Find the 8-BIT ARITHMETIC AND LOGIC group in Appendix C. The compare instruction is considered to be a logic instruction. Find the row labeled COMPARE 'CP'. The compare instruction in EXERCISE 1 is of type immediate so find the column labeled IMMED. What is the opcode for compare? In exercise 1 what value does n represent?

**5-14**

Can the contents of register B be compared to the contents of register A?

**5-15**

Turn to the JUMP GROUP in Appendix C. Earlier the UNCONDitional RELATIVE jump was examined. Now two new jumps (conditional jumps) are examined - jump relative if zero, and jump relative if non zero. What is the opcode if jump relative if zero? What is the opcode of jump relative if non-zero? How could you determine the mnemonic for jump if zero?

**5-16**

Write a program to HALT if any key with a key code is pressed except the [STEP] key. Write a program to HALT if the [GO] key is pressed. Write a program to halt only if the [STEP] key is pressed followed by pressing the minus [-] key. Test your answers by running your programs. Start thinking about this!

You may find the exercise difficult and you may not have the backgound. Build a combinations safe. A plus indicates a clockwise turn, and a minus indicates a negative turn. The safe will only open if you enter R14 L35 R7. If give the wrong combination an alarm goes off (for this problem a 1ØØØHZ tone).

**5-17**

What will be displayed the first time statement 4 in EXAMPLE 3 is executed?
What locations does the display buffer use?
How is the routine SCAN able to find the display buffer?

**5-18**

What is the opcode of the instruction LD HL,OUTBF?
The HL register pair is used as a pointer. What label and address does HL point to?
Are there any other pointers to OUTBF?
Why is HL pointing to OUTBF?

**5-19**

The description in section 5.5 under register states destroy AF, HL. Does this mean that these registers are useless after being used by HEX7SG routine.
What registers are destroyed by SCAN?

**5-20**

Why wasn't EXAMPLE 3 written as follows?

```
          ORG 1800H
          LD   IX,OUTBF
          LD   HL,OUTBF
  LOOP    CALL SCAN
          CALL HEX7SG
          JR   LOOP
```

How do you stop the program? Why are there two EQU statements? Add code to stop the program by pressing a key other than [RS] or [MONI]. What is the problem with exiting on a particular key code?

**5-21**

Why is EXAMPLE 4 of any value?
Where is the information to be displayed stored?
Change the program to display FEdCbA
Are there any instructions not previously explained in this program?
Why is B loaded with a 3?

**5-22**

How would you make each tone sound for .365 seconds? How would you make the lower sound last for .73 seconds and the higher sound last 1.46 seconds?
How would you add one more tone?

You will now vary paramenters (values) and listen to the results.

In statement 3 change 0 to at least three different numbers. In st atement 3 and 7 change 0 and 100H to at least three different values. In statement 4 try loading different

values into HL. In statement 8 try loading different values into HL.

How do you make two tones of equal time intervals? Read all of the information accompanying EXAMPLE 5 and the details of the TONE routine section 5.7. Pick two tones say 400 and 1000 cycles (Hertz) and the time interval (1 second).
If the frequency is already known, then to find C use the formula.

$$C = \left(\frac{200}{\text{Freq in KH}} - 10\right) / 3 = ?$$

For 400 Hertz

$$C = \left(\frac{200}{.400} - 10\right) /3 = 490/3 = ?$$

For 1000 Hertz

$$C = \left(\frac{200}{1.000} - 10\right) /3 = ?$$

Now compute the length of each sound at 400 Hertz

at 1000 Hertz

For equal time intervals of one second at 400 Hertz

at 1000 Hertz

In summary

| TONE | VALUE OF C | VALUE POF HL |
|---|---|---|
| 400 | | |
| 1000 | | |

5-23

Find the RESTART GROUP in Appendix C. What is the opcode for RST Ø ?
A restart instruction is a special form of a CALL instruction. RST Ø is equivalent to CALL ØØØØH. How many bytes in a restart instruction ? How many bytes is a CALL instruction ? Does a restart instruction save bytes? To location does RST Ø transfer control ? What happens to the contents of the old (next sequential location) program counter ?

Can the contents of the old program counter be accessed ?

5-24

The subtract instruction could also be used to clear A. For example SUB A,A subtract A from A would zero out the A register. Why wasn't SUB A,A used?

5-25

How would you test several PROMs and report which ROMs failed.

5-26

The Z-8Ø has another complement command NEG. This command will take the negative of the value in the A register. What is the opcode of the NEG?
How is the negative of plus two produced?

5-27

What does the CPI instruction do?
Writes a RAM test for a 4K RAM memory beginning at 2ØØØH.

## Answers

**5-1**

[; WE DON'T NEED ANY HELP]
[; WE DON'T NEED ANY HELP]
 ;
 ;
 ;

A semicolon does not have to be followed by text--comments.

**5-2**

[ORG 1900H]
[Your code would start at hexadecimal location C000. But the MPF-I as delivered does not have any memory at this location--so an ORG C000H might be a poor placement of the object code.]
[This is the space occupied by the monitor. Unless you are modifying or writing a new monitor locations, 0000H to 07FFH are to be avoided].

**5-3**

[11]
[3]
[no]
[no]
[yes]

**5-4**

[No, no entry under IMM EXT]
[LOC] [OBJ CODE] [STMT] [SOURCE STATEMENT]

**5-5**

[Look in Appendix C 16-BIT LOAD GROUP second table entry from the top.]
[IY <-- nn means that the immediate value nn will be loaded into the IX register.]

**5-6**

[Exchanges 'EX' AND 'EXX'. This group follows the 16 bit load instructions.]
[DDE3]
[a double opcode]
[180A-not 1810 the program counter uses hexadecimal values.]

**5-7**

[8-BIT LOAD GROUP 'LD']
[Ø6 n or Ø6 32]
[IMME.]
[REGISTER,B]
[2]
[5Ø is decimal, 32 is hexadecimal]
[The EXT. means extended and indicates a bigger immediate. 16 bits as opposed to 8 bits.]
[2]

**5-8**

[CALL, IMMED. EXT]
[CD]
[3]
[The location of the subroutine in EXERCISE 2 is the address of SCAN1]
[CD 24Ø6]

**5-9**

[18ØF]
[18ØC]
[1Ø]
[FF means jump back 1 byte. FE means jump back 2. So FB indicates a jump back of 5 (FF-1, FE-2, FD-3, FC-4, FB-5). A two byte backward jump will put the program counter at the beginning of the DJNZ instruction. 3 more bytes puts the program counter at the beginning of the CALL SCAN1 instruction-label HELFSEG.]

**5-10**

[18]
[11 decimal, B Hex]
[1813-B = 18Ø8 remember the program counter is at 1813]

**5-11**

[Enter BF in location 182Ø to 1825]
[My intials are RJB so in locations 182Ø to 1825 I would enter Ø3,B1,A7,ØØ,ØØ,ØØ]

**5-13**

| KEY | Ø | 1 | 5 | GO | MOVE | MONI |
|---|---|---|---|---|---|---|
| Code | ØØ | Ø1 | Ø5 | 12 | 1C | X |

[FE]
[13]

**5-14**

[ yes CP B ]

**5-15**

[28]
[2Ø]
[Turn to the second page of the JUMP GROUP. Under mnemonic find the JR commands. Under symbolic operation find if Z=O continue, if Z=1 PC <- PC + e. Remember if Z=Ø the zero flag was not set the result was not zero and one jump occurs. If Z=1 the contents of the PC is changed. Namely, jump if result is zero. The mnemonic is JR Z,e. In Exercise 1 e means the distance to jump.]

**5-17**

[Blanks ]
[19ØØ]
[SCAN uses IX to point to the display buffer]

**5-18**

[21]
[OUTBF, 19ØØ]
[Yes, index IX]
[We don't know yet. Statement 6 will reveal all]

**5-19**

[No, it means that HEX7SG wrote over the previous contents of AF and HL . Perhaps one should say alter rather than destroy.]
[AF, B, HL, AF', BC', DE']

**5-20**

[Because SCAN changes the contents of HL]
[Press Moni or RS]
[Two constants are needed. One for CALL SCAN and the other for CALL HEX7SG]

```
            ORG   18ØØH
            LD    IX,OUTBF
LOOP        CALL  SCAN
            CP    a key code;       This is the key code
            JR    EXIT;             That stops the program
            LD    HL,OUTBF
            CALL  HEX7SG
            JR    LOOP
```

[One key code will not be displayed]

**5-21**

[Because once the basic actions are understood, you can use this routine in a longer program to display results]
[Statements 16,17,18 locations 1900,1901,1902. The label is BYTE0]
[Change statement, 16,17 and 18 to

```
1900   BA   16   BYTEO   DEFB   OBAH
1901   DC   17           DEFB   ODCH
1902   FE   18           DEFB   OFEH
```

[Yes INC DE- increment DE adds one to the register pair DE]
[Statement 19 is also new. DEFS 6 define storage reserves a number of bytes, 6 in this case, in memory. This area is used as a display buffer.]
[So that the loop shown below

```
   LOOP    LD      A,(DE)
           CALL    HEX7SG
           INC     DE
           DJNZ    LOOP
```

will be executed three times . The first execution of the loop will convert the two digit (10) in BYTE 0 to display code and put the codes in OUTBF and OUTBF + 1. The next loop will convert the two digits (32) in BYTE 0+1 to display code and put the codes in OUTBF +2 and OUTBF +3. The final converts 54 and puts the result in OUTBF +4 and OUTBF +5.

**5-22**

[Change statements 4 and 7 to
statement 4 LD HL,060H
statement 7 LD HL,80H]
[Change statement 7 to statement 7 LD HL, 0FFH]
[After statement 8 add the code

```
LD   C,060H
LD   HL,0E0H
CALL TONE
```

[163]
[63]
[44 + 13 x 163] x 2 x 0.56 = 2423 micro sec]
[44 + 13 x 63] x 2 x 0.56 = 967 micro sec]

1/2423 x 0.000001 = 1/.002423 = 412 periods
1/.000967 = 1034 periods

| TONE | VALUE OF C | VALUE OF HL |
|---|---|---|
| 400 | 163 | 412 |
| 1000 | 63 | 1034 |

**5-23**

[C7] [1] [3] [YES] [0000H]
[Just as in the CALL instruction the program counter is saved on the stack]
[Yes if no other instructions that affect the stack are used, a RET (return) instruction will reload the program counter.]

**5-24**

[The XOR instruction always clears the carry flag, even when the operand isn't A (e.g. XOR B). The SUB A,A also clears the carry flag but it is easier to remember this fact when using XOR actually either instruction is equally good.]

**5-25**

[Test each ROM separately. When any ROM fails, save the address range of that ROM. When all ROMs have been tested, report the ranges of ROMs that failed.]

**5-26**

[There are two opcodes ED and 44. This is an extended instruction. This instruction is not present in the 8080/85 computers. Observe the number chart below:

```
Hex       Binary
02 00000010  plus two      +2
01 00000001  plus one      +1
00 00000000  zero           0
FF 11111111  negative one  -1
FE 11111110  negative two  -2
```

[Write the value of plus two in binary

00000010

Take the one's complement--toggle each bit

11111101

Add one

```
   11111101
  +       1
 ----------
   11111110
```

We have

```
        F  E
```

NEG gives a result which is one greater than one's complement and thus is called the two's complement.

**5-27**

[It decrements the contents of the BC register so that the JP PE,RAMT instruction can determine when all of memory has been tested. Remember BC contains the memory size. The CPI instruction also advances the memory pointer HL to the next location to be tested.]
[Only the first two instructions need to be changed. LD HL,1800H becomes LD HL,2000H and LD BC,800H becomes LD BC,1000H.]

# CHAPTER 6

## Useful Routines

Once a program is tested and debugged, any part, or all, of the program can be used as a subroutine in a larger program. You can build up a libary of useful routines by understanding how to use the programs presented in this chapter. Knowing how a program works permits you to tailor it to a specific application. Understanding a program also allows you to write a more powerful general subroutine--e.g., extending the range of a multiplication routine. All of the experiments referred to below are in the MPF-I Experiment Manual unless otherwise noted.

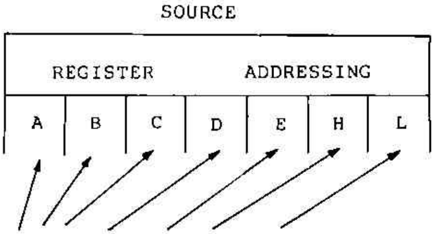
## Some basic principles—Applications of arithmetic and logical instructions

Turn to Experiment 2 Basic Applications of Arithmetic and Logic Operation Instructions in the MPF-I Experi-ment Manual. Read Section I, Theoretical Background. Some of the concepts presented in this section are for your review.

Adding is considered to be a fundamental process. You can add numbers rapidly because you have memorized the one hundred basic combinations such as: 3 + 4, Ø + 7, 8 + 9, and 9 + 8. The computer has been given a few rules also. The Z8Ø CPU instruction set allows either 8 bit adds (one byte) or 16 bit adds (two bytes). In 8 bit adds, the A register is always one of the numbers added (augend), and it also contains the result (sum).

Permissible 8-bit Adds

In the MPF-I User's Manual, turn to Appendix C. Find the 8-bit Arithmetic and Logic Chart. Find the row labeled ADD. The registers that can be added to A are given under Register Addressing (Fig. 6-1).




Any of these can be added to the A register

Fig 6-1

The Assembly language instructions are of the form

ADD A, r

where r is any one of the registers A,B,C,D,E,H,L. You can verify this by turning to second page of the 8-bit Arithmetic and Logical Group and looking at the first entry in the column labeled Mnemonic (fig. 6-2).

The permissible values of r (fig. 6-3) are listed in Comments column. To perform an add of two with registers, both the A register and the selected register (the r register) must be first loaded.

| Mnemonic |
| --- |
| ADD A,r |

Fig 6-2

| Comments | |
| --- | --- |
| r | Reg |
| 000 | B |
| 001 | C |
| 010 | D |
| 011 | E |
| 100 | H |
| 101 | L |
| 111 | A |

Permissible values of r

Fig 6-3

Exercise 6-1, 6-2

The value to be added to the A register may be accessed from memory by using the HL register pair as a pointer (fig. 6-4). The source is register indirect (REG. INDIR). This means that a register (or register pair) will point to the byte in memory to be used as the source.

| REG INDIR |
| --- |
| (HL) |

Fig 6-4

Exercise 6-3

Two other pointers to memory are permitted. Either index register IX or IY may point to the byte to be added to the accumulator--A register(fig.6-5). An offset of up to +127 or down to -128 is allowed with either index registers. The source is named INDEXED.

| INDEXED | |
|---|---|
| (IX+d) | (IY+D) |

Fig 6-5

Exercise 6-4

A constant may be added to the A register. The column labeled immediate is used to determine the hex code (fig. 6-6). The range of decimal numbers that can be used in a signed add is +127 and -128.

| IMMED |
|---|
| n |

Fig 6-6

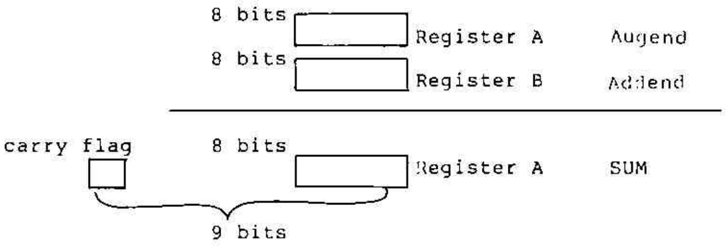
Exercise 6-5

If the result of an addition has to be contained in a byte and all the numbers were unsigned--essentially always positive, then the largest answer would be 255 (decimal)=FF (hexadecimal). Even more restrictive is the use of signed numbers. The leftmost bit is used for the sign of the number. Then only 7 bits are available for the size of the answer. The largest result would be 127, the smallest-128.

Exercise 6-6

Fortunately a method of extending the size (precision) of numbers used in addition has been provided. When- ever two numbers are added, the result is checked by the Z8Ø for a carry. If the two numbers didn't produce a carry, a flag called the carry flag is reset (cleared). If a carry is produced, then the carry flag is set. The carry flag adds an extra bit in the answer.

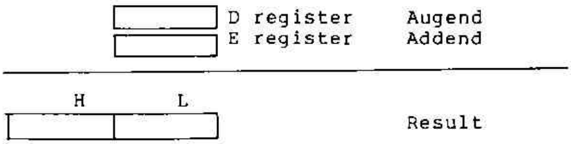
ADD A,B




Now an unsigned answer can be as large as 511 (decimal) = 1FF(hexadecimal). Proper use of the carry flag can extend both the size of unsigned and signed additions; the process is explained in the following example. The program shown below is the first example in section II. Example of Experiments under Experiment 2.

```
Statement        Source statement
   1             ORG 1800H
   2             LD    A,E
   3             ADD   A,D
   4             LD    L,A
   5             LD    A,0
   6             ADC   A,0
   7             LD    H,A
   8             RST   38H
```

Diagram



Statements 2 to 4

A conventional add of D and E with the 8 bit result in L.

Statement 5

The A register is zeroed out.

Statement 6

The add with carry, ADC, instruction adds the two operands, A and zero, and the carry flag together. The result is in A. The carry flag was set or reset by the ADD in Statement 3. The reason for the ADC A,Ø was to transfer the contents of the carry flag to the A register.

Exercise 6-7

Statement 7

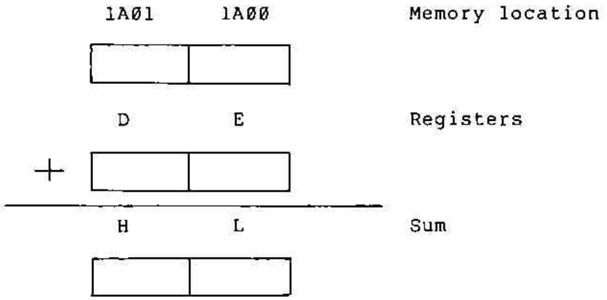
Transfer the carry (or no-carry) that was loaded into A into H.

Statement 8

The RST 38H instruction enters the monitor without executing the power-up code.

Exercise 6-8

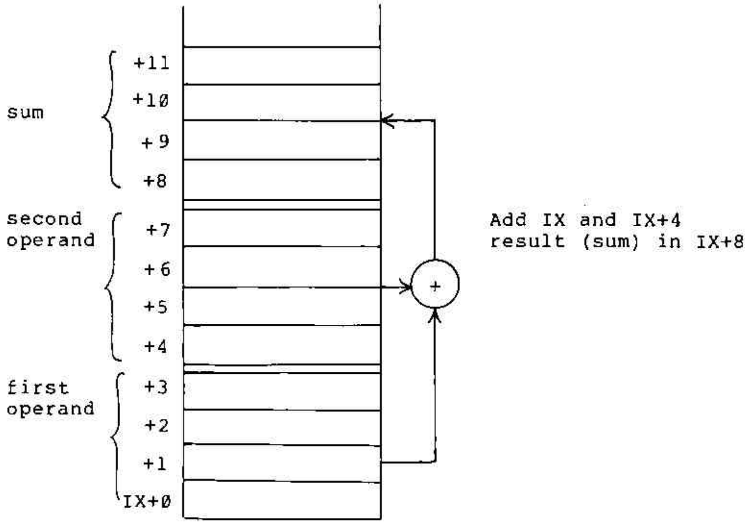
The second example (Example 2) under II. Example of Experiments (In Experiment 2 in the MPF-I Experiment Manual) can best be explained by a diagram.



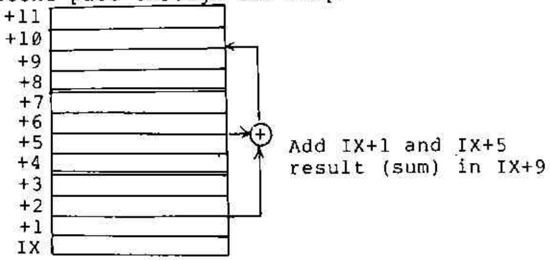

Exercise 6-9

Exercise 4 is also best understood by using a series of diagrams and a flowchart.

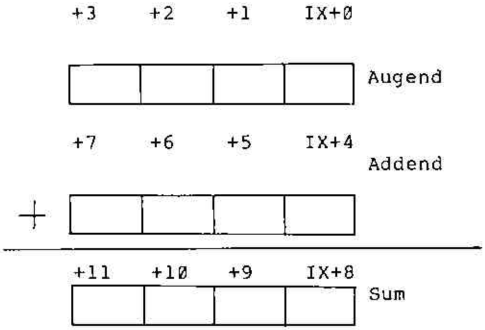
First pass through the loop



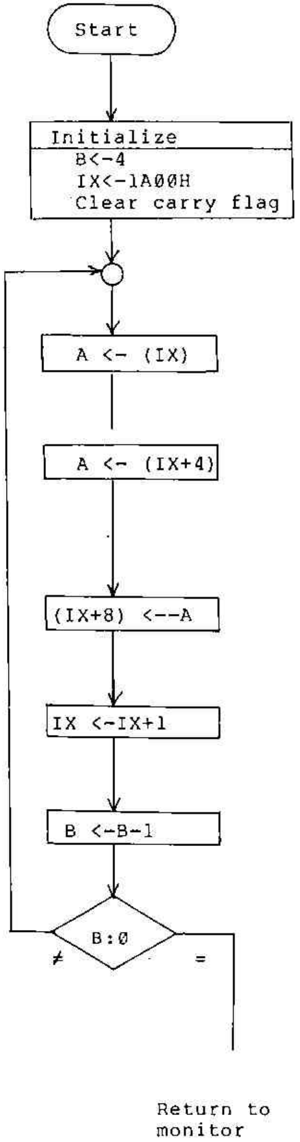
Second pass through the loop




Location of the operands



Flowchart




Instructions

```
ORG     1800H
LD      B,4
LD      IX,1A00H
AND     A
LD      A,(IX)
ADC     A,(IX+4)
LD      (IX+8),A
INC     IX
DEC     B
JP      NZ,LOOP
RST     38H
```

Study the charts, diagrams and the code. You should be able to understand how the program works.

Exercise 6-10

Exercise 6-11

Read the instructions in Example 5 (Experiment 2, MPF-I Experiment Manual).

The DAA stands for Decimal(ly) Adjust the Accumulator. Consider the problem below

```
  99
 +98
 ----
```

The result should be 197, if a decimal answer is desired. The computer will display the result of 9+8 as 16+1. To the computer 16 means produce a carry so put down a one and carry 1.

```
  1
  99
 +98
 ----
   1
```

Now 9+9+1 will be seen as 16+3. Put down a 3 and carry 1.

```
        99
       +98
 ---------------
  [1]   31
 Carry
```

For reference purpose, the right hex digit in a byte is called the right nibble and the left hex digit, the left nibble.

```
          |   3   |   1   |
  Left Nibble      Right Nibble
```

The carry bit is a flag altered by the add instruction. Another flag affected by the add instruction is the half carry flag. Whenever a carry is produced by adding the two right hex digits, a half carry flag is set. Adding 9+8 did produce a carry, so the half carry flag is set. The DAA instruction will add 6 if the left nibble is a hexadecimal number or if the half carry flag is set.

```
           99
           98          Half Carry
Carry [1]  31          [1]
            6          DAA instruction
           --
            7
```

Then if the left nibble is a hexadecimal number or if the carry flag is set then a 6 bit is added to the left nibble.

```
           99
          +98
          ---
           37
          +6
          ---
    [1]    97
Carry
```

Now you have the correct decimal result. Nibble is sometimes spelled Nybble.

Exercise 6-11b

Experiment 3 (MPF-I Experiment Manual)--more addition and subtraction.

Read Theoretical Background Section I.

Exercise 6-12

Read Theoretical Background Section 2,3, and 4.

Exercise 6-13

Perform II.1. in the II. Student Exercises.

Exercise 6-14

Perform II.2. in the Student Exercises. Read Exercise 6-15 first.

Exercise 6-15

Perform II.3. in the Student Exercises.

Exercise 6-16

Perform II.4. in the Student Exercises.

Exercise 6-17

Read and perform Experiment 3-1 in the Student Exercises.

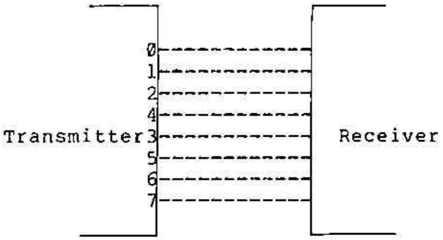
Exercise 6-18

Read and perform Experiment 3-2 in the Student Exercises.

You can use both ADD A,(nn) and ADC A,(nn).

Experiment 4: Branching and Looping

Read Theoretical Background part 1,2, and 3 in Experiment 4. By now you should understand the carry and zero flags. Parity will now be explained. Consider the circuit below




For transfer of information, you need lines 0 to 6. Line 7 is an unused spare. If a 3 was sent lines 0 and 1 would be high 11 (binary)=3 (decimal). If line 0 was open, then a two would be sent 10 (binary) =2 (decimal). How would the receiver know that line 0 is open? In the diagram above, there is no way of knowing.

A transmitter can be designed to count the number of set bits in each transmission on lines 0 to 6. Furthermore the transmitter can use line 7 to always make the total number of set bits in lines 0 to 7, odd or even. If an odd number of set bits is desired (even parity), then line 7 would be high when a three is sent. The byte would be 1000 0011. If a four is sent, line 7 is held low--0000 0100. A five gives 1000 0101. Bit 7 is used as the parity bit. A receiver can check parity by using either a fixed hardware design or software.

The transmitter and receiver are made to agree on whether even or odd parity will be used. A parity error results when a line is open, grounded, or shorted to another line. The receiver detects the parity error and informs the operator of unreliable transmission.

Parity can be tested by software by using one of the following logic commands AND, OR, XOR. ANDing the A register will test for even or odd parity and does not alter the contents of A. If an odd number of bits are set (on, high) in the A register, then the parity flag (P/V flag) is cleared (reset, zero). If an even number of bits are set, then the parity flag is set (on, high).

Exercise 6-19

Now read the remainder of I. Theoretical Background. How does one understand a new program? For example, the program loop in Section 5. The first thing you hope for is good documentation. Documentation consists of explanation in the form of paragraphs and comments given with most instructions. Many programmers "play computer". As they read through the program, they pretend that they are the computer and ask what is happening to the registers, the memory, and is data being sent to or received from external devices (peripherals). Restudy the program loop and play computer.

Exercise 6-20

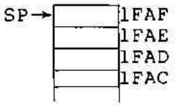
Experiment 5: Stack and Subroutines

Read about the stack which is discussed in section I. Theoretical Background. Be careful most of the instruction numbered (1) to (17) don't exist in the Z80 instruction set. They are used to demonstrate how PUSH and POP work. The program

```
        LD      SP,1FAFH
        PUSH    HL
        PUSH    AF
        POP     BC
        POP     DE
```

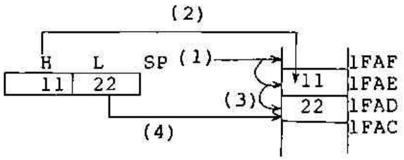
is shown below with drawings

LD SP,1FAFH




RAM Memory

PUSH HL

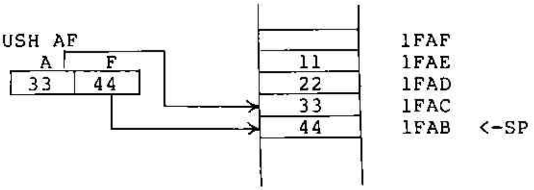



1) Decrement the stack pointer.

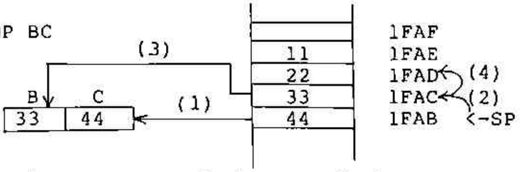
2) Contents of register H to the stack--H is not changed.

3) Decrement stack pointer.

4) Contents of register L to the stack--L is not changed.




POP BC



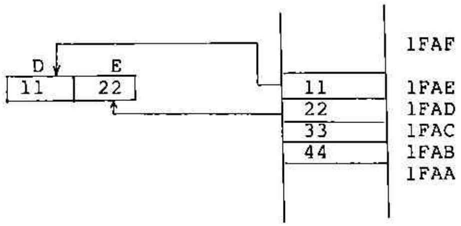

1) The contents of the top of the stack are loaded into register C.

2) Increment the stack pointer. Now the top of the stack is 1FAC.

3) The top of the stack is popped to register C.

4) Increment stack pointer.

POP DE




Read Section 2. Subroutine:

Exercie 6-21

BCD stands for Binary Coded Decimal. What this means is that the computations will be in a decimal form. This allows operating on decimal numbers (adding, subtracting, etc.) The reason for the DAA (Decimal Adjust the Accumulator) instruction at statement 12 is to insure a decimal result after each addition. Each time the computer adds, it produces a hexadecimal result which must be converted to decimal.

Read II. Example Experiment of the Experiment 5.
This experiment should read: Perform the following

BC → HL
DE → BC
HL → DE

using stack operations

Exercise 6-22

Experiment 6: Rotate Shift Instructions and Multiplication Routines

When a CPU chip is designed, the designer decides what features to incorporate. There is a limited amount of space (Real Estate) in a chip. The instruction set must be choosen very carefully. Until recently chips containing multiply instruction were expensive and sometimes very specialized. How can a useful CPU be built that doesn't contain a hardware multiply instruction? A hardware multiply means that the multiply is accomplished by circuits built into the CPU chip. A multiply is a series of actions. You can multiply by using a series of instructions other than the multiply command. A very essential instruction is the ability to shift and/or rotate. Read Section 1. under the Theoretical Background. This section will introduce you to the rotate and shift instruction group. Don't try to memorize the instructions in this group. There are too many of them.

Exercise 6-23

Read Sections 2. Binary Multiplication: to 5. Program flowchart.

These are not easy sections. The object is to show you how to multiply by shifting, bit testing, and adding. Read these sections several times.

Follow II. Example Experiments:

Exercise 6-24

Experiment 7: Binary Division Routine

Read 1. Binary division by hand calculation. If you are overwhelmed (snowed) by the explanation, you have a binary choice. You may accept that the division method works and proceed to 2. Division Program Design or read the explanation below.

The problem is really

```
10100 ) 11101101  ==>  20 ) 237
```

The first step is

```
10100 ) 11101101 <-dividend
                   divisor
```

shift divisor until it becomes smaller then the dividend

```
   00001        quotient
  ) 11101101
    10100
```

Then put a one in the quotient.

Now subtract

```
   00001
  ) 11101101
    10100
    ----------
     1001  <-- New dividend
```

Bring down the next digit to the new dividend

```
10100 )11101101
        10100
        10011
```

Test the divisor

```
        00001
10100 ) 11101101
        10100
         10011
         10100
```

No, dividend is too small. So put a zero in the quotient and bring down the next digit

```
        000010
10100  )11101101
        10100 ↓
         100110
```

Now divisor is smaller than the dividend. Put a one in the quotient and subtract.

```
        0000101
10100  )11101101
        10100
         100110
          10100  Subtract
          10010  New Quotient
```

Bring down the next digit.

```
          0000101
10100   )11101101
          10100
           100110
            10100
            100101
```

Divisor is smaller than dividend. Put a one in the quotient and subtract.

```
          00001011     <-quotient(17) divisor 20)237   divisor
10100   )11101101     <-dividend(237)              11    quotient
  ↑        100110                                   20
divisor     10100                                   37
  (20)      100101                                  20
             10100                                  17    remainder
             10001     <-remainder(17)
```

Read 2. Division Program Design

Exercise 6-25

Experiment 8: Binary-to-BCD Conversion Program

Read only 1. Methods of binary-to-BCD conversion. A sample conversion--Convert

00010011 (binary) to decimal. The correct answer is 19 (decimal).

| Number to be converted | BCD area |
|---|---|
| 0001 0011 | 0000 0000 |
| Shift most significant into | |
| carry, add, carry, and double | |
| BCD number | 0000 0000 |
| | 0000 0000 |
| | +0 |
| | 0000 0000 |
| | |
| 0001 0011 | 0000 0000 |
| Add, carry, and double | 0000 0000 |
| BCD number | +0 |
| | 0000 0000 |
| | |
| 0001 0011 | 0000 0000 |
| Add, carry, and double | 0000 0000 |
| BCD number | +0 |
| | 0000 0000 |
| | |
| 0001 0011 | 0000 0000 |
| BCD number | 0000 0000 |
| Add, carry, and double | +1 |
| | 0000 0001 |
| | |
| 0001 0011 | 0000 0001 |
| Add, carry, and double | 0000 0001 |
| BCD number | +0 |
| | 0000 0010 |
| | |
| 0001 0011 | 0000 0010 |
| Add, carry, and double | 0000 0010 |
| BCD number | +0 |
| | 0000 0100 |
| | |
| 0001 0011 | 0000 0100 |
| Add, carry, and double | 0000 0100 |
| BCD number | +1 |
| | 0000 1001 |
| | |
| 0001 0011 | 0000 1001 |
| Add, carry, and double | 0000 1001 |
| BCD number | +1 |
| | 0001 0011 |
| | |
| Execute a DAA instruction because | |
| a half carry occured | 0001 0011 |
| Add 6 = 0110 (binary) | + 0110 |
| | 0001 1001 |

The answer is 1 9

A second conversion converts 1111 1111 (binary) to 255 (decimal). The purpose of the shift into carry flag is so that the selected bit can be added to the shifted result. Each add (double) and shift in carry will be shown as one step

```
Number to be converted     Zeroed out BCD number
 1  1  1  1  1  1  1  1   0000 0000 0000 0000
 ↑  ↑  ↑  ↑  ↑  ↑  ↑  ↑
(1)(2)(3)(4)(5)(6)(7)(8)                          decimal
Shift and add bit (1)     0000 0000 0000 0001       1
Shift and add bit (2)     0000 0000 0000 0011       3
Shift and add bit (3)     0000 0000 0000 0111       7
Shift and add bit (4)     0000 0000 0000 1111
BCD adjust                               0110
                          -------------------
                          0000 0000 0001 0101      15

Shift and add bit (5)     0000 0000 0010 1011
BCD adjust                               0110
                          -------------------
                          0000 0000 0011 0001      31

Shift and add bit (6)     0000 0000 0110 0011      63
Shift and add bit (7)     0000 0000 1100 0111
BCD adjust                          0110
                          -------------------
                          0000 0001 0010 0111     127

Shift and add bit (8)     0000 0010 0100 1111
BCD adjust                               0110
                          -------------------
                          0000 0010 0101 0101     255
```

The answer 255 is correct. Read 2. Assembly Language Programming Technique

Exercise 6-26

Example Experiments

Exercise 6-27

Experiment 9: BCD-to-Binary Conversion Program

The basic method of hand conversion is given in 2. Principle of the checking process (3) under Theoretical Background. As you can see by dividing the number to be converted repeatedly by 2 and saving the remainder, a rather easy conversion is obtained.

Exercise 6-28

Now a method of dividing over and over by 2 is needed. Shifting a binary number to the right always divides by 2 with a remainder of 1 or 0. Shifting a BCD number to the right will give an incorrect result in two bit positions in each byte. Read 1. and 2. under Theoretical Background. The shifting problem is explained.

Conversion from BCD to binary is rather straight foreward. The program must: (1) divide by 2 (2) save the remainder (3) correct two bits in each byte of the BCD number,and (4) have two loop controls--one for the number of BCD bytes and a second one total number of divides respectively.

Exercise 6-29

Experiment 10: Square-Root Program

Square root has never been considered one of the easier methematical operations. Years ago, the only easy method was to use square-root tables. Various other methods existed for those who were interested in expanding mental effort--slide rules, logarithms (again tables), and a hand method which consisted of doubling dividing, and subtracting. The hand method given in this experiment is easier than one taught in schools before calculators. Binary numbers lends themselfs to square root computations. Read 1. Calculating square roots of binary numbers by hand. Try very carefully to follow the processes.

The square root of larger numbers can be calculated by enlarging X,Y, and R. The square root routine in section 2. expands the size of number whose square root is to be found and the size of the answer. First read only up to the program. Now you will match the program with the flowchart.

Statement 7: LD A,B

The original data is not stored in register A and C but in BC. So statement 7 loads B into A.

Statement 8: LD B,16

The original data to be shifted is contained in two registers A and C. The 16-bit data is shifted two bits at a time so the shift count would be 8. The fractional part of the answer is 8 bits, thus 8 more shifts of 2 bits each time are required. The total shifts, tests and subroutines are 16.

Statements 9-11

These statements will zero out the X area, HL, and the R area, DE. HL ← DE ← Ø

Statements 12-13

Statement 12 subtracts (N) 4ØH from the contents of the accumulator. On the first pass, A will contain the upper part of the original data. Statement 13 subtracts R,(DE) from X,(HL).

```
HLA ← HLA←DEN
XY    XY   RP
```

Statement 14

If DE is less or equal to HL, then the results of the subtraction performed in statement 13 are to be kept. In this case, the carry flag will not be set and control is transferred to location SQ1 statement 17. If DE is greater than HL, the number in HLA, XY needs to be restored.

Exercise 6-3Ø

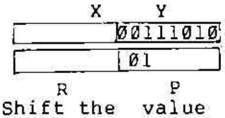
The square root of 81 and 16 are whole numbers (integers). For a more interesting case, consider the square root of 58. The square root of 58 is approximately 7.615. In binary numbers, a bit represents twice as much as the bit to the right and half as much the bit to the left. In 111, the middle bit represents value of 2. The left bit is equal to 4 decimal and the right bit is equal to 1 decimal.

```
1     1     1
4   + 2   + 1 = 7 decimal
```

What is one-half of one? One-half (1/2). Going to the right of the binary point gives .1 (binary) which equals .5 decimal. The next position to the right is one half of 1/2 or 1/4 (.25)

.Ø1 (binary)=.25 (decimal) To represent .75 use two bits .11= .5 (decimal)+ .25 (decimal)= .75 (decimal) To obtain decimal (fractional) results in taking square root, continue the shifting process beyond the integer part of the number.

Interger result only




58 (decimal)

Shift the value in XY 4 times (2 bits each). The result R will be ØØØØ Ø111 (7 decimal).

Fractional result

Shift the value in XY four more times. Now bits representing

```
1/2  .5
1/4  .25
1/8  .125
1/16 .Ø625
```

have been used, the answer is: Ø111.1ØØ1

Under these conditions the carry flag will be set -- the jump instruction will not break the sequential flow and statement 15 is executed next.

Statements 15 and 16

The original values subtracted from A and HL are added back in. Thus the original number is restored except the carry flag will be set. Remember this!

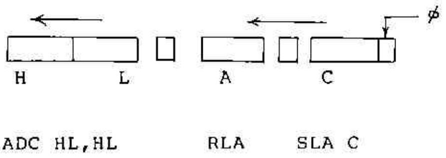
STATEMENTS 17 to 19

The carry flag will be shifted into R ( register D & E ). If RP ( register D, E and a constant are smaller than or equal to XY ( register H, L, A, and C ) then the carry flag should be one ( set ). If RP was greater than XY then the carry flag should be zero ( reset ).

However, the subtraction in statement 13 has left the carry flag in the opposite condition, thus statement 17 complements the carry flag. Statement 18 and 19 rotate D and E one place to the left. The carry flag enters the rightmost bit of E.

STATEMENTS 2Ø to 26

The first shift to the left of H, L, A and C is performed by statement 21 to 23. The second left shift by statement 24 to 26. Statement 21 -- shift C to the left one bit and put a zero in rightmost bit. Statement 22 -- rotate A to left and receive the carry from C. Statement 23 -- shift HL to the left by doubling the register pair HL and accept the carry from A by adding with carry.




Statement 28 -- Loop back 15 times ( a total of 16 passes ) to SQO.

## Questions of Exercises

6-1 Which of the following instructions are not allowed -- give the reason?

a) ADD A,B    b)ADD E,A    c)ADD A,HL
c) ADD A,A    d)ADD AC,DH

6-2 On the second page of 8-BIT ARITHMETIC AND LOGICAL GROUP is a column titled symbolic operation, explain the meaning of A <= A+r for the ADD A,r instruction.

6-3 (a) Using the first page of the 8-BIT ARITHMETIC AND LOGICAL GROUP find the opcode for adding the memory location (pointed to by HL) to the A register? Using the second page of this same group locate the row containing the symbolic operation for register indirect.
(b) What is the symbolic operation?
(c) What is the mnemonic in the same row? Find the intersection with the column labled opcode.
(d) What is the opcode ?
(e) What is hexadecimal equivalent of 10000110?

6-4 Refer to the first page of the 8-BIT ARITHMETIC AND LOGICAL instructions.
(a) What is the opcode for ADD A, (IX+4)?
(b) What is the significance of the +4?
(c) How does +4 show up in the hexadecimal codes DD 86 d?

6-5 (a) Write the mnemonic (assembly language code) for an add 3 to the A register.
(b) Write the mnemonic for adding -4 to A.
(c) The hexadecimal code for ADD A, 3 is C603. Can you guess what the hex code for ADD A, -4 is?

6-6 If A contains 74 hexadecimal and B contains BF hexadecimal will the instruction ADD A,B add a) a negative number to a positive number b) two negative numbers C) two positive numbers. What do you think the rightmost bit would be called?

6-7 The add with carry instruction comes in all the same flavors as the ADD command. Use the information in Appendix C 8-BIT ARITHMETIC AND LOGIC GROUP both pages to answer the following questions. Fill in the blank entries, [ ] below.

| Instruction | object code (hexadecimal) |
| --- | --- |
| ADC A,D | [ (a) ] |
| ADC A,(IX+d) | [ (b) ] |
| ADC A,(IX+4) | [ (c) ] |
| [ (d) ] | FD 8E 25 |
| [ (e) ] | FD 8E FD |

(f) The mnemonic for add with carry is given as ADC A,s (see second page of 8-bit ARITHMETIC AND LOGICAL GROUP). What does the s mean?

6-8 Execute the first exercise (I) under example of experments (of Experiment 2 of MPF-I Experiment Manual). Fill in the chart shown in this section.

6-9 (a) What is accomplished by the instructions

```
LD A,(1A00H)
ADD A,E
LD L,A
```

Show your answer by using a diagram.

(b) See Experiment 2 (II. 2) What is accomplished by the instructions

```
LD A,(1A01H)
ADC A,D
LD H,A
```

Again show your answer using a diagram.

(c) Will above code always give a correct result?

(d) Using another method add two 16 bit numbers. The operands are in the locations 1A00 and 1A01 as before but the result (sum) is stored in HL.

A new instruction was used that requires knowledge of 16-bit arithmetic. Turn to Appendix C 16-BIT ARITHMETIC. The second page of this section shows the Mnemonic ADD HL,ss in the first row. The Comments column shows ss to be any one of BC,DE,HL, SP. Thus ADD HL, DE is a legal instruction. The Symbolic Operation column shows HL is added to ss and the result is placed in HL. When ss is DE the operation is HL <- HL+DE

(e) Load and execute exercise 2.

6-10 Add comments to each statement below

(a) LD B,4

(b) LD IX, 1A00H

(c) AND A

(d) LD A,(IX)

(e) ADC A,(IX+4)

(f) LD (IX+8),A

(g) INC IX

(h) DEC B

(i) JP NZ,LOOP

(j) What two instructions could be replaced by one instruction?

(k) What is the replacement?

(l) Load and execute exercise 4.

6-11 Expand example 4 to add a 64 bit number.

Expand example 4 to add a 128 bit number.

6-11b Perform 5. in Example of Experiments.

6-12 In example 3-1 convert all the numbers to base ten decimal. Show your answers.

(a)

| HEX | 7F | AD | AC | 2E |
|---|---|---|---|---|
| DEC | | | | |

(b) Now check the results of the addition 7F+AD=? and subtraction 7F-AD=? Are the answers correct?

(c) In Example 3-2 what adjustments would have to be made if the leftmost addition results in a carry?

(d) What is the significance of a set carry bit after a subtract operation?

(e) How many borrows occured in Example 3-2?

6-13

(a) Fill in the names of operands in the boxes below. Use Sum, Augend, Addend.

(b) Again enter the names of the operands in the boxes below. Use Subtrahend, Minuend, and Difference.

(c) Study again the flowchart for addition. Note that the decision box at the second step from the end ◇, can cause a repeat of 5 steps. Each repeat is called a pass. The page after the flowchart shows what events occur on the first pass. The diagram may be a little hard to read at first. What is the first event?
Second event?
Third event?
Fourth event?
Fifth event?

(d) The top part of the next page shows the events of the second pass. The results of the third and final pass are shown at the bottom of this page. The complete program is shown at the end of this section. Fill in the values of the registers. The carry flag and memory locations for each step.

| INSTRUCTION | REGISTERS | | | | FLAG |
|---|---|---|---|---|---|
| | A | B | (IX) | (IY) | Z |
| ADD3 XOR A<br>LD B,3 | | | | | |
| ADDLP LD A,(IX)<br>ADC A,(IY)<br>LD (IX),A<br>INC IX<br>INC IY<br>DJNZ ADDLP | | | | | |
| ADDLP LD A,(IX)<br>ADC A,(IY)<br>LD (IX),A<br>INC IX<br>INC IY<br>DJNZ ADDLP | | | | | |
| ADDLP LD A,(IX)<br>ADC A,(IY)<br>LD (IX),A<br>INC IX<br>INC IY<br>DJNZ ADDLP<br>RET | | | | | |

6-14

Show the object code and location counter in the listing below. Assume the program starts at location 1800H.

```
                                       EXP3
 LOC    OBJ CODE M STMT SOURCE STATEMENT

                                  ORG      1800H
                      7           LD       B,3
                      8           XOR      A
                      9   ADDLP   LD       A,(IX)
                      10          ADC      A,(IY)
                      11          LD       (IX),A
                      12          INC      IX
                      13          INC      IY
                      14          DJNZ     ADDLP
                      15          RST      38H
```

6-15

To execute the 3-BYTE ADDITION PROGRAM. You must first have IX and IY point to the data. There are two ways to do this. What are they?

6-16

To perform the subtraction statement 10 was changed from ADC A, (IY) to SBC A, (IY). (a). What was the code for ADC A, (IY)? (b). What is new code for SBC A, (IY)? (c). Why is the third byte of each command zero?

6-17

(a) In adjusting to five byte data how many lines of the program changed?
(b) What changes were made?

6-18

(a) When is it correct to call the rightmost bit of the flag register a carry flag?
(b) When is it correct to call the rightmost bit a borrow flag?
(c) Read and perform Experiment 3-2 in Experiment 3 of the MPF-I Experiment Manual. you can use both ADD A,(nn) and ADC A,(nn).

6-19

(a) What is the parity of the bytes given below?
    0110 1100
    0100 0000
    0111 1111
    0100 0001

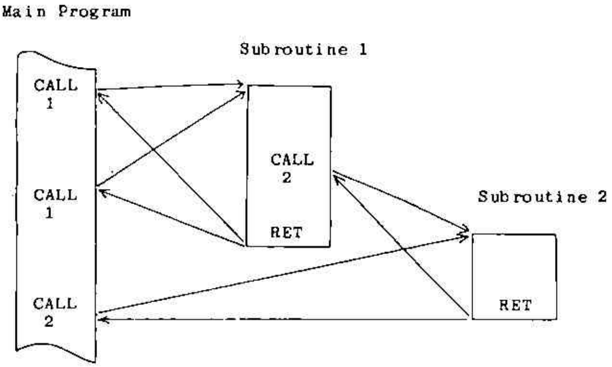
(b) In the bytes below what would be the setting (state) of the parity bit (7) to have even parity?
    0110 1100
    0100 0000
    0111 1111
    0100 0001

6-20

Example Experiments of Experiment 4 (MPF-I Experiment Manual)

Exercise 1 Follow the instructions--Before executing the program add comments to each instruction.

6-21 Label the order of the actions in the diagram below




6-22

(2) Explain this program statement by statement. Note after shifting left four bits with method shown below could result in the loss of data if the original number is greater than 15 decimal.

| | | | |
|---|---|---|---|
| Ex. | | 1 1 Ø Ø | number is 12 (decimal) |
| | 1 1 Ø Ø | Ø Ø Ø Ø | shifting gives no data lost |
| | | 1 1 1 1 | number is 15 (decimal) |
| | 1 1 1 1 | Ø Ø Ø Ø | no data lost |
| | 1 | Ø Ø Ø Ø | number is 16 (decimal) |
| | Ø Ø Ø Ø | Ø Ø Ø Ø | data lost, bit was shifted out of the register by adding |

6-23

Find the ROTATE AND SHIFT GROUP in Appendix C in the MPF-I User's Manual. On the second page of this group find the column labelled Symbolic Operation.

Except for the last two operations RLD & RRD , all of the instructions operate on 8 bits and the carry flag, CY.

(a) What is the real difference between instructions starting with R (rotate) and starting with S (Shift)?

(b) Again, look at the diagrams for the rotate instructions, the bit shifted out of the byte is transferred into the carry flag and in some cases the bit is also transferred to the other end of the byte. How are these two cases separated by the assembler?

(c) Draw the symbolic operation for

RLA
RRA
RLCA
RRCA

6-24

II. Sample Experiments

(a) 1. Draw a diagram showing how the shift is performed

(b) 4. Comment on each line of the program show how it works

6-25

Perform the exercises given in Illustrations of Experiments.

6-26

Study the sample program EX001 LISTING

STATEMENTS 15 through 20 clear the BCD area. This is the area where the result will be developed.

The contents of a particular register is loaded into all the BCD bytes.

(a) What register is used?

(b) What statement zeros out A?

(c) What statement puts zeros into the BCD bytes (one for each loop)?

(d) What are the statement numbers in the loop that zeros out the BCD bytes?

(e) How many passes will be made ?

(f) At what statement was B loaded with the number of bytes to zero out ?

(g) STATEMENTS 22 to 27 computes the number of shifts to be made.

If the binary number consists of 3 bytes how many shifts into the carry flag must be made ?

(h) Assume D = 3 number of binary bytes. Statement 23 will load this value into the A register. What do statements 24 to 26 do ?

(i) What is happening at statement 27 ?

STATEMENTS 3Ø to 35 will shift all the binary bytes one to the left and leave the carry flag with the highest order bit.

(j) What is the address of the first byte to be shifted ?

(k) What register pair points to memory when the ROTATE LEFT (RL) command is executed ?

(l) How is the starting addres for each series of shifts loaded into HL ?

(m) What statement numbers are contained in the loop that adjusts all the BCD bytes each time a new binary bit is available in the carry flag ?

(n) How many passes will be made through the loop ?

(o) What do statements 47 and 48 decide ?

6-27

Example Experiments

Perform experiments

6-28

(a) convert the decimal number 9 to binary. Show the process.

(b) Convert the decimal number 492 to binary show the process.

6-29

The BCD-to-Binary conversion program given in section 3 will now be analyzed.

STATEMENTS 11 TO 17 divide the BCD number by 2. The result must be tested for adjustment of bits 7 and 3.

(a) How many bytes will be rotated to the right by one place ?

(b) STATEMENTS 18 to 24 check the two potentially incorrect bits.
Why is bit 7 being tested in statement 19 ?

(c) What is statement 21 doing ?

(d) What is the other bit position to be tested ?

(e) Statement 24 corrects what ?

(f) Discuss Statements 26 to 29 ?

(g) The STATEMENTS 32 to 35 rotate the bit that was shifted out of the BCD numbers into the high order byte and rotate all the binary bytes to the right. How many binary are there ?

(h) Discuss STATEMENTS 37 to 38.

6-30

Now that you have read how to hand calculate square root, solve the problems below.

Compute the square root of 16 (decimal). Show the results of each text, subtraction, and shift.

## Answers to Exercises

6-1 [b) Operands are in the wrong order for the assembler
correct instruction is ADD A,E
c) Can't add the 16-bit register pair HL to the 8
bit register A answer must fit in an 8 bit byte.
e) can't pair A and C or D and H.]

6-2 [The value of r is added to the contents of the A
register. The result, sum, is put into A.]

6-3 a. [86]
b. [A<-- A+(HL)]
c. [ADD A, (HL)]
d. [10000110]
e. [86]

6-4 a.[DD86 the index instructions have an extended
opcode.]
b.[The memory location referenced will be four more
than the value of IX. For example if IX = 7000 then
memory location 7000 is referenced.]
c.[The 04 replaces the d.]

6-5 a.[ADD A,3]
b.[ADD A,-4]
c.[C6FC]

6-6 [a) 74 hexadecimal=01110100min binary so A is a
positive number. The leftmost bit is zero. This is
called the most significant bit MSB. The number
in B BF hexadecimal = 10111111 is a negative
number, the MSB is 1.]

[The least significant bit LBS. It is also bit number 0.]

6-7 a.[8A]
b.[DD 8E d]
c.[DD 8E 04]
d.[ADC A, (IY+25H)]
e.[ADC A, (IY-3)]
f.[See Comments s is any of r,n,(HL),(IX+d), (IY+d).
Also under comments r is given as any of
B,C,D,E,H,L,A.]

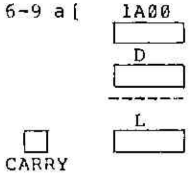
6-8

| Preset Value | | Result of Program Execution | | | | |
|---|---|---|---|---|---|---|
| Register | | Register | Flag | | | |
| D | E | HL | Sign | Zero | P/V | Carry |
| 5AH | A6H | Ø1ØØ | Depends on when sampled | | | |
| 46H | 77H | ØØBD | | | | |

ADD

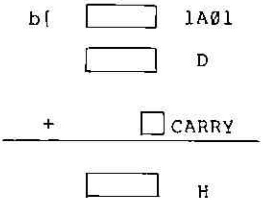
```
LOC    OBJ CODE M STMT SOURCE STATEMENT

18ØØ                1          ORG     18ØØH
18ØØ   7B           2          LD      A,E
18Ø1   82           3          ADD     A,D
18Ø2   6F           4          LD      L,A
18Ø3   3EØØ         5          LD      A,Ø
18Ø5   CEØØ         6          ADC     A,Ø
18Ø7   67           7          LD      H,A
18Ø8   FF           8          RST     38H
```

6-9 a [




]

b [




]

c [ No, if the values in 1AØØH to 1AØ1H and DE are large, the result (sum) will be 17 bits in size. ]

d [
```
LD    L,(1AØØH)
LD    H,(1AØ1H)
ADD   HL,DE
```
]

```
                                    ADD16BIT
 e  LOC   OBJ CODE M STMT SOURCE STATEMENT

    1800               1          ORG     1800H
    1800   3A001A      2          LD      A,(1A00H)
    1803   83          3          ADD     A,E
    1804   6F          4          LD      L,A
    1805   3A011A      5          LD      A,(1A01H)
    1808   8A          6          ADC     A,D
    1809   67          7          LD      H,A
    180A   FF          8          RST     38H
```

RESULTS

| 1A01 | 1A00 | DE | HL | |
|---|---|---|---|---|
| 00 | 01 | 0004 | 0005 | |
| 01 | 01 | 0703 | 0804 | Zero flag set |

6-10 a. [The number of passes through the loop 4 is loaded into the B register.]

b. [Load the base (starting) value in the index register IX.]

c. [Clear the carry flag.]

d. [Load the first operand into the A register. (augend) ]

e. [add the second operand to A; the result (sum) is in A. A <- (IX) + (IX + 4)]

f. [Store the current sum at IX +8.]

g. [advance IX to point to the next set of operands and sum. ]

h. [B <- B-1 ]

i. [If the result of decrementing B is non-zero then loop back to LOOP. ]

```
 1. LOC OBJ CODE M STMT SOURCE STATEMENT

    1800               1          ORG     1800H
    1800   0604        2          LD      B,4
    1802   DD21001A    3          LD      IX,1A00H
    1806   A7          4          AND     A
    1807   DD7E00      5  LOOP    LD      A,(IX)
    180A   DD8E04      6          ADC     A,(IX+4)
    180D   DD7708      7          LD      (IX+8),A
    1810   DD23        8          INC     IX
    1812   05          9          DEC     B
    1813   C20718     10          JP      NZ,LOOP
    1816   FF         11          RST     38H
```

FOR ADD

| 1A03H-1A00H | 1A07H-1A04H | 1A0BH-1A08H | FLAG REG |
|---|---|---|---|
| 3B712345 | 8FFDAA10 | CB6ECD55 | 42 |
| FFFFFFFF | FFFFFFFF | FFFFFFFF | 43 |

6-11

```
[ OLD                     NEW
  LD  B,4                 LD  B,8
  ADC A,(IX+4)            ADC A,(IX+8)
  LD  (IX+8),4            LD  (IX+16),A ]

[ OLD                     NEW
  LD  B,4                 LD  B,16
  ADC A,(IX+4)            ADC A,(IX+16)
  LD  (IX+8),A            LD  (IX+32),A ]
```

6-11b

FOR SUBTRACT

| 1A03H-1A00H | 1A07H-1A04H | 1A0BH-1A08H | FLAG REG |
|---|---|---|---|
| 8FFDAA10 | 3B712345 | 548C86CB | 42 |
| FFFFFFFF | FFFFFFFF | 00000000 | 42 |

```
                                    SUB4B
 LOC    OBJ CODE M STMT SOURCE STATEMENT

1800                  1           ORG     1800H
1800    0604          2           LD      B,4
1802    DD21001A      3           LD      IX,1A00H
1806    A7            4           AND     A
1807    DD7E00        5   LOOP    LD      A,(IX)
180A    DD9E04        6           SBC     A,(IX+4)
180D    DD7708        7           LD      (IX+8),A
1810    DD23          8           INC     IX
1812    05            9           DEC     B
1813    C20718       10           JP      NZ,LOOP
1816    FF           11           RST     38H
```

FOR ADD & DAA

| 1A03H-1A00H | 1A07H-1A04H | 1A0BH-1A08H | FLAG REG |
|---|---|---|---|
| 12345678 | 87654321 | 99999999 | 42 |
| 35868794 | 44556699 | 80425493 | 42 |

```
                                    ADD4BDA
 LOC    OBJ CODE M STMT SOURCE STATEMENT

1800                  1           ORG     1800H
1800    0604          2           LD      B,4
1802    DD21001A      3           LD      IX,1A00H
1806    A7            4           AND     A
1807    DD7E00        5   LOOP    LD      A,(IX)
180A    DD8E04        6           ADC     A,(IX+4)
180D    27            7           DAA
180E    DD7708        8           LD      (IX+8),A
1811    DD23          9           INC     IX
1813    05           10           DEC     B
1814    C20718       11           JP      NZ,LOOP
1817    FF           12           RST     38H
```

6-12 a[

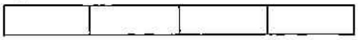
| HEX | 7F | AD | 12C | 2E |
|---|---|---|---|---|
| DEC | | | | |

]

b[ Yes ]

c Four bytes would have to be reserved for the answer (not three). The carry would be placed in the highest order byte of the answer.

Carry



High order byte (Most significant digit) — Low order byte (Least significant byte)

d[ A borrow has occurred. ]

e[ 2 ]

6-13 a[ AUGEND

\+ ADDEND

SUM ]

b[ MINUEND

− SUBTRAHEND

DIFFERENCE ]

c[ LD A,(IX) Load the accumulator with the contents of the memory location pointed to by the IX index register. ]

[ ADC A,(IY) Add to the accumulator the contents of the memory location pointed to by the IY index register. ]

[ LD (IX),A Store the accumulator away in the memory location pointed to by the IX index register. ]

[ INC IX Advance by one the IX register. ]

[ INC IY Advance by one the IY register. ]

d

| INSTRUCTION | | | REGISTERS<br>A B (IX) (IY) | | FLAG |
|---|---|---|---|---|---|
| ADD3 | XOR | A | ? ? BD AC | | Ø |
| | LD | B,3 | ? 3 BD AC | | Ø |
| ADDLP | LD | A,(IX) | BD 3 BD AC | | Ø |
| | ADC | A,(IY) | 69 3 BD AC | | 1 |
| | LD | (IX),A | 69 3 69 AC | | 1 |
| | INC | IX | 69 3 7C AC | | 1 |
| | INC | IY | 69 3 7C 65 | | 1 |
| | DJNZ | ADDLP | 69 2 7C 65 | | 1 |
| ADDLP | LD | A,(IX) | 7C 2 7C 65 | | 1 |
| | ADC | A,(IY) | E3 2 7C 65 | | Ø |
| | LD | (IX),A | E3 2 E3 65 | | Ø |
| | INC | IX | E3 2 6A 65 | | Ø |
| | INC | IY | E3 2 6A 4B | | Ø |
| | DJNZ | ADDLP | E3 1 6A 4B | | Ø |
| ADDLP | LD | A,(IX) | 6A 1 6A 4B | | Ø |
| | ADC | A,(IY) | B5 1 6A 4B | | Ø |
| | LD | (IX),A | B5 1 B5 4B | | Ø |
| | INC | IX | B5 1 ? 4B | | Ø |
| | INC | IY | B5 1 ? ? | | Ø |
| | DJNZ | ADDLP | B5 1 ? ? | | Ø |
| | RET | | | | |

6-14

```
                                      EXP3
 LOC    OBJ CODE M STMT SOURCE STATEMENT

1800                           ORG      1800H
1800    0603         7         LD       B,3
1802    AF           8         XOR      A
1803    DD7E00       9  ADDLP  LD       A,(IX)
1806    FD8E00      10         ADC      A,(IY)
1809    DD7700      11         LD       (IX),A
180C    DD23        12         INC      IX
180E    FD23        13         INC      IY
1810    10F1        14         DJNZ     ADDLP
1812    FF          15         RST      38H
```

6-15 [ One--Change the code. Between statements 8

(LD B,3) and statement 9 (LD A,(IX) insert

```
LD IX,1900H
LD IY,1A00H
```

Two--Load IX and IY from the keyboard.
Press REG, IX, 1, 9, 0, 0
Press REG, IY, 1, A, 0, 0

The test data must also be loaded by entering

ADDR, 1, 9, 0, 0, DATA, B, D, +, 7, C, +, 6, A

ADDR, 1, A, 0, 0, DATA, A, C, +, 6, 5, +, 4, B

To run the table data--first set--replace the test data by

ADDR, 1, 9, 0, 0, DATA, 6, 5, +, 3, 8, +, 9, 7
ADDR, 1, A, 0, 0, DATA, D, F, +, C, E, +, A, B ]

6-16 a[ FD 8E 00 ]
b[ FD 96 00 ]
c[ A displacement of zero was used--in effect, there is no displacement. ]

6-17 a[ 1 ]
b[ Statement 8 became LD B,5 ]

6-18 a[ When addition or incrementation is performed. It is not incorrect to call this flag a carry flag when subtraction is performed. ]
b[ Only when subtraction is performed. ]

c
```
XOR   A
LD    A,(1820H)
ADD   A,(1823H)
LD    (1826H),A
LD    A,(1821H)
ADD   A,(1824H)
LD    (1827H),A
LD    A,(1822H)
ADD   A,(1825H)
LD    (1828H),A
```

6-19a[ Even ]
[ Odd ]
[ Odd ]
[ Even ]

b[ Reset (clear) ]
[ Set (on) ]
[ Set (on) ]
[ Reset (clear) ]

6-20

```
[        ORG 1800H         ;Program code starts at 1800.
         LD HL,1900H       ;The HL register pair will point
                           ;to the memory location in which
                           ;a byte is to be placed.
         LD B,20H          ;B is the loop counter which is
                           ;used with DJNZ, 20 passes will
                           ;be made through the loop.
LOOP     LD (HL),A         ;The current value in A will
                           ;be stored at the location
                           ;pointed to by HL.
         INC HL            ;Advance the memory pointer so
                           ;that the next sequential memory
                           ;location can receive the contents
                           ;of A.
         DJNZ LOOP         ;Decrement the loop counter B
                           ;and return to LOOP if B is non
                           ;zero.
         RST 38H           ;Enter the monitor program.
```

Answers to Experimental results (1), (2), (3) under exercise 1 of II. Example Experiments ( Experiment 4, MPF-I Experiment Manual )

(1) 1900H to 191FH are zeroed out. 1920H is unchanged.
(2) Now locations 1900H to 191FH contain 55H
1920H is unchanged.
(3) Locations 1900H to 19FFH contain 64H.
Remember loading zero in B and using DJNZ for loop control will give 256 passes. DJNZ will first decrement the value in B then test 00H-1=FFH (255 decimal).

2. Trace this program in your mind--play computer (Trace the program in Exercise 2. Nested loops under II. Example Experiments of MPF-I Experiment Manual).

Results:
(1) Memory locations

| 1900-190F | 1910-191F | ... | 19E0-19EF | 19F0-19FF |
|---|---|---|---|---|
| 00 | 01 | | 0E | 0F |

Did you get the same results?

(2) Revised changes are

LD HL,1900H in place of LD HL,19FFH and
INC HL instead of DEC HL

Test your program.

3. Read MPF-I Experiment Manual, Experiment 4, II. Example Experiments, Exercise 3. first.

Since DEC BC doesn't set flags, the JR NZ,LOOP will be useless. Between DEC BC and JR NZ,LOOP insert

```
LD A,B
OR C
```

If any bit is set, the OR command will reset the zero flag indicating a non-zero result.

4. Read MPF-I Experriment Manual, Experiment 4, II. Example Experiments, Exercise 4, (1)

(1) Comment for each statement

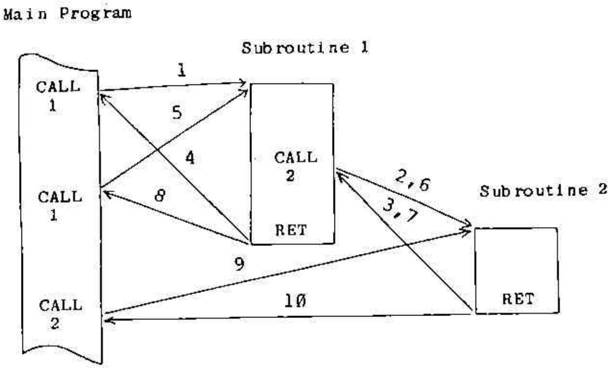
```
[     ORG 1800H      ;Program begins at 1800H
      LD HL,1B00H    ;First base address from which
                     ;data will be transferred.
      LD DE,1A00H    ;First destination address for
                     ;data movement.
LOOP  LD A,(HL)      ;These two instructions move one
                     ;byte
      LD (DE),A      ;From a source address pointed to
                     ;by HL to a destination address
                     ;pointed to by DE.
```

```
      CP 0FFH        ;After each byte is transferred,
                     ;the A register will still contain
                     ;a copy of the byte.  Compare FF
                     ;against the contents of the A
                     ;register.  If A contains zero,
                     ;set the zero flag.
      JR Z,EXIT      ;If the compare instruction
                     ;found a zero in the A register,
                     ;then a jump to EXIT will be made.
      INC HL         ;Continue here if A was not equal
                     ;to 0FFH.  Advanced the source
                     ;pointer to prepare for the
                     ;next move.
      INC DE         ;Advance the destination pointer.
      JR LOOP        ;Make another pass through the
                     ;loop
EXIT  RST 38H        ;Transfer control to the monitor.
```

(2) Comment on each instruction

```
     ORG 18ØØH        ;Start program code at 18ØØH
LOOP LD A,(HL)        ;The current contents of the
                      ;memory location pointed to
                      ;by HL is loaded into A.
     NEG              ;Gives a two's complement of A
     LD (HL),A        ;Return complement value of A
                      ;to memory.
     INC HL           ;Advance memory pointer
     AND A            ;Clear the carry flag to get
                      ;a correct result in the next
                      ;subtraction.
     SBC HL,DE        ;If HL less than DE, then
                      ;the zero flag is not set.
     ADD HL,DE        ;Restore the data at HL to
                      ;its original state.
     JR NZ,LOOP       ;If the result of the
                      ;SBC HL,DE was non-zero (HL
                      ;still less than DE), then
                      ;transfer control to LOOP.
```

6-21




6-22

ANSWER (1)

| | | |
|---|---|---|
| PUSH | HL | E5 |
| PUSH | DE | D5 |
| PUSH | BC | C5 |
| POP | HL | E1 |
| POP | BC | C1 |
| POP | DE | D1 |

```
(2) 1            ORG    1800H     ;Set location counter to zero.
    2            LD     B,21H     ;Loop 21 times.
    3            LD     HL,1A00H  ;First location to be shifted.
    4  LOOP1     PUSH   BC        ;Save BC on stack because it
                                  ;will  be altered by the  inner
                                  ;loop (LOOP2).
    5            LD     A,(HL)    ;Load memory byte to shifted
                                  ;4 places into A.
    6            LD     B,4       ;Number of adds (shifts) is 4.
    7  LOOP2     ADD    A,A       ;Each add will shift value in A
                                  ;left one place.
    8            DJNZ   LOOP2     ;Loop control--4 passes (loops).
    9            LD     (HL),A    ;Return  shifted  value  to
                                  ;memory.
    10           INC    HL        ;Advance memory pointer.
    11           POP    BC        ;Relocated value of BC that was
                                  ;preserved on the stack by
                                  ;statement 4.
    12           DJNZ   LOOP1     ;Have 21 numbers been shifted?
                                  ;No, loop back to LOOP1.
    13           HALT             ;Yes. Quit.
```

(3) Change statement 11 to read ADC A,(HL)

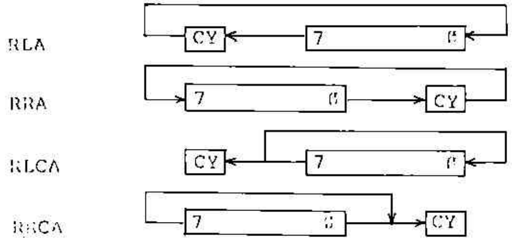
```
LD     HL,1A80H
LD     DE,1A00H
LD     IX,1A80H
LD     B,8
CALL   MADD
```

6-23

a [In shift instructions the bit shift out of either the 8 bit byte or the carry flag is not rotated around to opposite end. It will be lost]

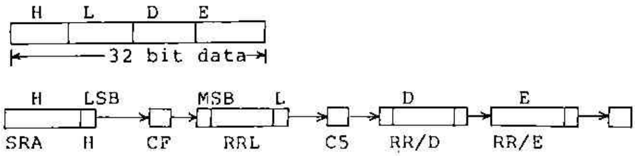
b [The presence of a C in the third position indicatees a transfer both into the carry flag and the opposite end of the byte.]

c use MPF-I manual page C-17




6-24

a [




Note the carry flag has been drawn in several places
for convenience this is the same carry flag. ]

[MULTIPLY X 2

```
            ORG     1810H
            SLA     E
            RL      D
            RL      L
            RL      H
            RST     38H      ]
```

2. [ANSWER

```
            ORG     1830H
            LD      B,5
    LOOP2   PUSH    BC
            LD      HL,1A00H
            LD      B,4
            AND     A
    LOOP1   RL      (HL)
            INC     HL
            DJNZ    LOOP1
            POP     BC
            DJNZ    LOOP2    ]
```

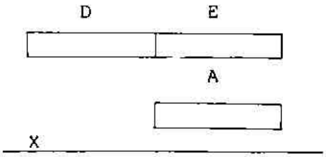
3. [ANSWER

```
                ORG     1800H
                LD      B,4
        LOOP2   PUSH    BC
                XOR     A
                LD      HL,1A00H
                LD      B,4
        LOOP1   RLD     (HL)
                INC     HL
                DJNZ    LOOP1
                POP     BC
                DJNZ    LOOP 2   ]
```

b 4. [

```
MPYS    LD      BC,800H  ; Load B with 8 thus shifting
                         ; the value in A 8 times.
                         ; Zero out C
        LD      H,C      ; Zero out the H register
        LD      L,C      ; Zero out the L register
M1      ADD     HL,HL    ; Shift the sum left one place
        RLA              ; Rotate the most significant
                         ; bit of A into the carry flag
        JR      NC,M2    ; Test if carry is set means
                         ; that an add should occur
        ADD     HL,DE    ; Add if carry set
        ADC     A,C      ; Put bit shifted out of A back
                         ; into opposite end of A
M2      DJNZ    M1       ; Are there more bits to be
                         ; tested in A
        RST     38H      ; Return to the monitor
```

This program is different from the theoretical background problem in only one respect. The theoretical background problem is an 8 bit by 8 bit multiplication and in this example a 16 bit number is multiplied by an 8 bit number.




So done

II. Answer to II. 5 is in Answer to Experiment 6

6-25 See answers in 6-24

6-26

a [ A ]

b [ 15 an exclusive OR of A will clear A and the carry flag. ]

c [ 18 ]

d [ 18 to 20 ]

e [ The number of passes equals the value in B ]

f [16 the D register contains the number of bytes in BCD area. ]

g [ 24 ]

h [ Each statement doubles the value of A. The final result is 8 * A =24 if D = 3 ]

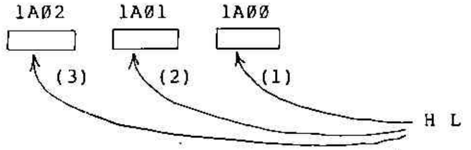
i [ Register C will hold the number of shifts. ]

j [1AØØH]

k [The HL register pair -- see statement 33]

l [ Statement 17 loads H with 1A the value of H never changes. Statement 31 zeros out the L register ]

In summary:




The numbers (1), (2), (3) are pass numbers.

m STATEMENTS 37 to 45 double the number; add the carry (obtained from shifting the binary number to be converted and then decimally adjust all the BCD bytes.

n [ 4Ø to 45 ]

o [ The B register controls the number of passes. B is loaded with D which has the number of BCD bytes. ]

[ Statement 47 decrements the bit count and statement 48 decides whether all of the bits in the BCD number have been processed. ]

Trace the program again, it is a good practice.

6-28

a

```
     9
  ÷  2
  ------
     4
  ÷  2
  ------
     2
  ÷  2
  ------
     1
  ÷  2
  ------
     0 -------> 1   0   0   1
                8   4   2   1        8 + 1 = 9
```

b

```
    492
  ÷   2
  ------
    246
  ÷   2
  ------
    123
  ÷   2
  ------
     61
  ÷   2
  ------
     30
  ÷   2
  ------
     15
  ÷   2
  ------
      7
  ÷   2
  ------
      3
  ÷   2
  ------
      1
  ÷   2
  ------
      0 ------>  1    1    1    1    1    0    1    1    0
               256  128   64   32   16    8    4    2    1

               256 + 128 + 64 + 16 + 4 + 2 = 492
```

a [ 5 -- statement 12 loads the B register with 5 and the loop at statements 15 to 17 is controlled by the DJNZ statement. ]

b [ A shift of a bit into this position doesn't divide the higher digit by 2. The digit is worth 8Ø not 5Ø. ]

c [ Applying a correction of 3Ø. 8Ø - 3Ø = 5Ø this statement is only executed if bit 7 is set.]

d [ Bit 3. Statement 22 a shift of a bit into this position provides an 8 instead of a 5. ]

e [ Bit position 3. 8 - 3 = 5 ]

f [ The conected byte is stored away statement 26. HL is decrement to point to the next lower byte statement 27. The contents of the carry flag are restored for use in the next potential shift statement 28. Now the value in B is tested to determine if more bytes are to be shifted. Statement 29 , a total of 5 bytes are to be processed on each pass. See statement 12 ]

g [ 4. See statement 32. Statement 35 forces a loop back to SHR4 if all the shifts have not occurred.]

h [ These statements are responsible for determing if all the BC bits have shifted right. Statement 9 sets the count to 32 and statement 37 decrements the by one. Statement 38 tests count. If register C -- the shift count register - is non zero, a jump back to DBLP is executed.]

6-3Ø

```
[        X        Y
                  ØØØ1 ØØØØ = 16 (decimal)
                  Ø1ØØ ØØØØ
         R        P
```

I have used a carat, ^ , to show the end of the original value. ]

RP is greater than XY so in the hand method you avoid subtraction. The computer has to subtract to determine the relationship between RP and XY. If RP is greater than XY restore the original result. Shift XY two places to the left. Do not change RP.

| X | Y |
|---|---|
| | Ø1 ØØ ØØ^ØØ |
| | Ø1 |
| R | P |

marks the end of the original value

Now RP equals XY so subtract R P from XY. Shift R one place to the left and set the least significant bit (rightmost bit) of R. Shift XY two places to the left.

| X | Y |
|---|---|
| | 00 00 00 00 |
| 1 | 01 |
| R | P |

RP is greater than XY; Shift XY left two places, shift R left one place.

| X | Y |
|---|---|
| | 00 00 00 00 |
| 10 | 01 |
| R | P |

RP is greater than XY; Shift XY left two places, shift R left one place.

| X | Y |
|---|---|
| 00 00 00 00 | 00 00 00 00 |
| 1 00 | 01 |
| R | P |

These were 8 bits in the original number (in Y). Four left shifts, 2 places each time, completes the processing. The answer is in R, 100 (binary) = 4 (decimal), which is the square root of 16 (decimal).

# EXPERIMENTS

### Experiment 2

Answer to 1 under II. Example of Experiments in the MPF-I Experiment Manual, Experiment 2--Basic Applications of Arithmetic and Logic Operation Instructions.

```
 LOC    OBJ CODE M STMT SOURCE STATEMENT

1800                 1           ORG      1800H
1800    7B           2           LD       A,E
1801    82           3           ADD      A,D
1802    6F           4           LD       L,A
1803    3E00         5           LD       A,0
1805    CE00         6           ADC      A,0
1807    67           7           LD       H,A
1808    FF           8           RST      38H
```

Ansers to 2. under II. Example of Experiments of Experiment 2 of the MPF-I Experiment Manual.

```
                                 ADD16BIT
 LOC    OBJ CODE M STMT SOURCE STATEMENT

1800                 1           ORG      1800H
1800    3A001A       2           LD       A,(1A00H)
1803    83           3           ADD      A,E
1804    6F           4           LD       L,A
1805    3A011A       5           LD       A,(1A01H)
1808    8A           6           ADC      A,D
1809    67           7           LD       H,A
180A    FF           8           RST      38H
```

```
   RESULTS
1A01     1A00      DE      HL
 00       01      0004    0005
 01       01      0703    0804   Zero flag set
```

3. Change ADC A,(IX+4) to SBC A,(IX+4)

4.

```
                                       ADD4B
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                     1             ORG      1800H
1800    0604             2             LD       B,4
1802    DD21001A         3             LD       IX,1A00H
1806    A7               4             AND      A
1807    DD7E00           5  LOOP       LD       A,(IX)
180A    DD8E04           6             ADC      A,(IX+4)
180D    DD7708           7             LD       (IX+8),A
1810    DD23             8             INC      IX
1812    05               9             DEC      B
1813    C20718          10             JP       NZ,LOOP
1816    FF              11             RST      38H
```

FOR ADD

| 1A03H-1A00H | 1A07H-1A04H | 1A0BH-1A08H | FLAG REG |
|---|---|---|---|
| 3B712345 | 8FFDAA10 | CB6ECD55 | 42 |
| FFFFFFFF | FFFFFFFF | FFFFFFFE | 43 |

5. FOR SUBTRACT

```
                                       SUB4B
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                     1             ORG      1800H
1800    0604             2             LD       B,4
1802    DD21001A         3             LD       IX,1A00H
1806    A7               4             AND      A
1807    DD7E00           5  LOOP       LD       A,(IX)
180A    DD9E04           6             SBC      A,(IX+4)
180D    DD7708           7             LD       (IX+8),A
1810    DD23             8             INC      IX
1812    05               9             DEC      B
1813    C20718          10             JP       NZ,LOOP
1816    FF              11             RST      38H
```

| 1A03H-1A00H | 1A07H-1A04H | 1A0BH-1A08H | FLAG REG |
|---|---|---|---|
| 8FFDAA10 | 3B712345 | 548C86CB | 42 |
| FFFFFFFF | FFFFFFFF | 00000000 | 42 |

FOR ADD & DAA

```
                                   ADD4BDA
LOC     OBJ CODE M STMT SOURCE STATEMENT


1800                    1             ORG      1800H
1800     0604           2             LD       B,4
1802     DD21001A       3             LD       IX,1A00H
1806     A7             4             AND      A
1807     DD7E00         5   LOOP      LD       A,(IX)
180A     DD8E04         6             ADC      A,(IX+4)
180D     27             7             DAA
180E     DD7708         8             LD       (IX+8),A
1811     DD23           9             INC      IX
1813     05            10             DEC      B
1814     C20718        11             JP       NZ,LOOP
1817     FF            12             RST      38H
```

| 1A03H-1A00H | 1A07H-1A04H | 1A0BH-1A08H | FLAG REG |
|---|---|---|---|
| 12345678 | 87654321 | 99999999 | 42 |
| 35868794 | 44556699 | 80425493 | 42 |

**Experiment 3**

Answer to 2. under Student Exercises: of Experiment 3, in the MPF-I Experiment Manual.

```
LOC     OBJ CODE M STMT SOURCE STATEMENT


1800                    1             ORG      1800H
1800     0603           2             LD       B,3
1802     AF             3             XOR      A
1803     DD7E00         4   ADDLP     LD       A,(IX)
1806     FD8E00         5             ADC      A,(IY)
1809     DD7700         6             LD       (IX),A
180C     DD23           7             INC      IX
180E     FD23           8             INC      IY
1810     10F1           9             DJNZ     ADDLP
1812     FF            10             RST      38H
```

| Augend | Addend | Answer | Flags |
|---|---|---|---|
| 1902-1900 | 1A02-1A00 | 1902-1900 | |
| 793865H | ABCEDFH | 250744H | 31 |
| 009543H | AB1236H | A6A779H | A8 |
| 954717H | 003390H | 957AA7H | 80 |

3.

```
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                  1           ORG      1800H
1800    AF            2           XOR      A
1801    0603          3           LD       B,3
1803    DD7E00        4   SUBLP   LD       A,(IX)
1806    FD9E00        5           SBC      A,(IY)
1809    DD7700        6           LD       (IX),A
180C    DD23          7           INC      IX
180E    FD23          8           INC      IY
1810    10F1          9           DJNZ     SUBLP
1812    FF           10           RST      38H
```

| Minuend | Subtrahend | Answer | Flags |
|---|---|---|---|
| 1902-1900 | 1A02-1A00 | 1902-1900 | |
| 683147H | 336700H | 34CA47H | 22 |
| 5935ABH | 5877FFH | 00BDACH | 42 |
| 049677H | F65B79H | 0E3AFEH | 1B |

**Experiment 4**

Answer to Experiment 4, MPF-I Experiment Manual

1.

```
                                  EXP4
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                  1           ORG      1800H
1800    210019        2           LD       HL,1900H
1803    0620          3           LD       B,20H
1805    77            4   LOOP    LD       (HL),A
1806    23            5           INC      HL
1807    10FC          6           DJNZ     LOOP
1809    FF            7           RST      38H
```

2.

```
                                  EXP401
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                  1           ORG      1800H
1800    21FF19        2           LD       HL,19FFH
1803    0E0F          3           LD       C,0FH
1805    0610          4   LOOP2   LD       B,10H
1807    71            5   LOOP1   LD       (HL),C
1808    2B            6           DEC      HL
1809    10FC          7           DJNZ     LOOP1
180B    0D            8           DEC      C
180C    C20518        9           JP       NZ,LOOP2
180F    FF           10           RST      38H
```

3.

```
                                     EXP402
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                  1         ORG     1800H
1800    018001        2         LD      BC,0180H
1803    218018        3         LD      HL,1880H
1806    36AA          4  LOOP   LD      (HL),0AAH
1808    23            5         INC     HL
1809    0B            6         DEC     BC
180A    78            7         LD      A,B
180B    B7            8         OR      A
180C    20F8          9         JR      NZ,LOOP
```

4.

```
                                     EXP403
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                  1         ORG     1800H
1800    21001B        2         LD      HL,1B00H
1803    11001A        3         LD      DE,1A00H
1806    7E            4  LOOP   LD      A,(HL)
1807    12            5         LD      (DE),A
1808    FEFF          6         CP      0FFH
180A    2804          7         JR      Z,EXIT
180C    23            8         INC     HL
180D    13            9         INC     DE
180E    18F6         10         JR      LOOP
1810    FF           11  EXIT   RST     38H
```

5.

```
                                     EXP404
 LOC    OBJ CODE M STMT SOURCE STATEMENT


1800                  1         ORG     1800H
1800    7E            2  LOOP   LD      A,(HL)
1801    ED44          3         NEG
1803    77            4         LD      (HL),A
1804    23            5         INC     HL
1805    A7            6         AND     A
1806    ED52          7         SBC     HL,DE
1808    19            8         ADC     HL,DE
1809    20F5          9         JR      NZ,LOOP
```

**Experiment 5**

II. (2)

```
1           ORG    1800H     ;Set location counter to zero.
2           LD     B,21H     ;Loop 21 times.
3           LD     HL,1A00H  ;First location to be shifted.
4  LOOP1    PUSH   BC        ;Save BC on stack because it
                             ;will  be altered by the  inner
                             ;loop (LOOP2).
5           LD     A,(HL)    ;Load memory byte to shifted
                             ;4 places into A.
6           LD     B,4       ;Number of adds (shifts) is 4.
7  LOOP2    ADD    A,A       ;Each add will shift value in A
                             ;left one place.
8           DJNZ   LOOP2     ;Loop control--4 passes (loops).
9           LD     (HL),A    ;Return  shifted  value  to
                             ;memory.
10          INC    HL        ;Advance memory pointer.
11          POP    BC        ;Relocated value of BC that was
                             ;preserved on the stack by
                             ;statement 4.
12          DJNZ   LOOP1     ;Have 21 numbers been shifted?
                             ;No, loop back to LOOP1.
13          HALT             ;Yes. Quit.
```

(3) Change statement 11 to read ADC A,(HL)

```
        LD     HL,1A00H
        LD     DE,1A08H
        LD     IX,1A00H
        LD     B,8
        CALL   MADD
```

(4) For subtraction, change

```
ADC    A,(HL) to
SBC    A,(HL)
```

For multi-byte binary addition/subtraction, delete the DAA command at statement.

(5) Subroutine to complement HL

```
HLCOMP LD   A,H
       NEG
       LD   H,A
       LD   A,L
       NEG
       LD   L,A
```

Subroutine to complement IX and IY

```
IXCOMP PUSH IX
       POP  HL
       CALL HLCOMP
       PUSH HL
       POP  IX
IYCOMP PUSH IY
       POP  HL
       CALL HLCOMP
       PUSH HL
       POP  IX
```

(6) The method will be to 2's complement DE, then add DE to IY.

```
       PUSH DE
       POP  HL
       CALL HLCOMP
       PUSH HL
       POP  DE
       ADD  IY,DE
```

**Experiment 6**

II. Example Experiments:

5. This program is most easily solved by studying the register assignments and flowchart in the previous section on binary multiplication. The only difference is the size of the multiplicand, multiplier, and answer (product).

```
     32 bits (4 bytes)   Multiplicand

  x  32 bits (4 bytes)   Multiplier

     64 bits (8 bytes)   Product
```

The essential steps are

1) Clear the product area 8 bytes.
2) Initialize the shift counter - 32 shifts of the multiplicand, multiplier, and product.
3) Save the shift counter by PUSHing it onto the stack.
4) Shift the product area left one bit.
5) Shift the multiplier left into carrry.
6) Test the carry flag. If set fall through to 7). If zero, transfer to step 8).
7) Add the multiplicand to the product.
8) POP the shift counter into BC.
9) Test B--Done? Yes--Exit step 10) No--step 3)
10) Return to the monitor.

```
                   ;CLEAR THE PRODUCT AREA


         LD        HL,1A08H;First location

         LD        B,8       ;Byte count
         LD        A,0
CLEAR    LD        (HL),0    ;(HL)<-0
         INC       HL        ;Advance to next memory
                             ;location.
         DJNZ      CLEAR     ;More locations to be
                             ;cleared.

                   ;INITIALIZE SHIFT COUNTER

         LD        B,32      ;Total number of shifts.
SHFTAGANPUSH       BC        ;Save shift count


                   ;SHIFT PRODUCT AREA

         LD        B,8       ;Shift the product (8 bytes)
         XOR       A         ;Clear carry
         LD        HL,1A08H;First location
PRODSHFTRL         (HL)      ;Shift
         INC       HL        ;Advance to next memory
                             ;location
         DJNZ      PRODSHFT;More bytes to shift?


                   ;SHIFT THE MULTIPLIER

         LD        B,4       ;Total number of shifts
                             ;(4 bytes).
         LD        HL,1A04H;First location
PLYERSHFRL         (HL)      ;Shift
         INC       HL        ;Advance to next memory
                             ;location
         DJNZ      PLYERSHF;More bytes to shift?
```

```
        ;DO WE ADD?

JP      NC,SHIFT32;Jump if no carry
                  ;(no add).

        ;ADD

LD      B,4       ;Add 4 bytes
XOR     A         ;Clear carry
LD      HL,1A00H;Start of multiplicand
LD      IX,1A08H;Start of product
FIRSTADDLD      A,(HL)   ;Load multiplicand
ADC     A,(IX)    ;Add product
INC     HL        ;Advance multiplicand
                  ;pointer
INC     IX        ;Advance product pointer
DJNZ    FIRSTADD;More bytes to be added.

        ;ADD CARRIES IN UPPER PART OF PRODUCT

LD      B,4
SECNDADDLD      A,0
ADC     A,(IX)
LD      (IX),A
DJNZ    SECNDADD

        ;CHECK SHIFT COUNTER

SHIFT32 POP     BC        ;Restore BC
        DJNZ    SHFTAGAN;More shifts?
        RST     0FFH
```

The answer given above is one solution. You may wish to use more registers and improve the code.

**Experiment 7**

II. 2.

| [Dividend | Divisor | Answer | Remainders | Check |
|---|---|---|---|---|
| 8686H | 0020 | 0434 | 0006 | 00 |
| FFFF | 0005 | 5555 | 0000 | 10 |
| 5A48 | 0142 | 0047 | 00FA | 00 |
| 0H | 0142 | 0000 | 0000 | 40 |
| 1234H | 0H | FFFF | 1234 | bC |

3.
[The key to modifying the division routine in part one is to realized that the dividend is being shifted bit by bit out of BC. Also that the result is being shifted bit by bit in DE. This problem requests a 32 bit result, a 16 bit register and 16 bit fractional result. The division is still a 16 bit number (division) divided into a 16 bit number (dividend). Make the following changes:

```
Statement 12 change LD A,16 TO
                    LD A,32
```

After statement 12 add

```
        LD IX,1A00H
```

Replace statements 15 to 17 with

```
        RL (IX)
        RL (IX+1)
        RL (IX+2)
        RL (IX+3)
```

The dividend is in locations 1A00 and 1A01. The integer result will be locations 1A00 and 1A01. The 16 bit fractional result will be in locations 1A02 and 1A03. The jump relative instruction at statement 27 will have to be adjusted to jump the proper distance to DV0.

4.

```
        LD      D,(1A01H)

        LD      E,(1A00H)

        LD      B,(1A04H)

        LD      C,(1A05H)

        CALL    DIVI16

        LD      (1A01H),H

        LD      (1A00H),L
```

5.

```
  ;CLEAR LOCATIONS 1A04-1A07

        XOR     A
        LD      B,4
        LD      HL,1A04H
CLEAR   LD      (HL),A
        INC     HL
        DJNZ    CLEAR

  ;SET IX TO POINT TO DIVIDEND

        LD      IX,1A00H

  ;SET AND PRESERVE SHIFT COUNT

        LD      A,32
        PUSH    AF

  ;ROTATE DIVIDEND INTO TEST AREA
   ROTATE RESULT BITS INTO LEAST SIGNIFICANT BIT
   OF DIVIDEND AREA

DV0     RL      (IX)
        RL      (IX+1)
        RL      (IX+2)
        RL      (IX+3)

  ;SHIFT TEST VALUE

        LD      B,4
        LD      IY,1A04H
SHIFTTSTRL      (IY)
        INC     IY
        DJNZ    SHIFTTST

  ;SUBTRACT THE DIVISOR

        XOR     A
        LD      B,4
        LD      HL,1A20H
        LD      IY,1A04H
DIVISOR LD      A,(IY)
        SBC     A,(HL)
        INC     HL
        INC     IY
        DJNZ    DIVISOR

  ;WAS THE DIVISOR LARGER THAN THE TEST VALUE?
   NO. JUMP TO DV1

        JR      NC,DV1
```

```
  ;DIVISOR LARGER THAN TEST VALUE.  ADD DIVISOR
   BACK IN

        XOR     A
        LD      B,4
        LD      HL,1A20H;Divisor is at locations
                        ;1A20-1A23H
        LD      IY,1A04H
        LD      A,(IY)
RESTORE ADC     A,(HL)
        INC     HL
        INC     IY
        DJNZ    RESTORE

  ;ADJUST CARRY FLAG

DV1     CCF

  ;TEST FOR ALL 32 SHIFTS

        POP     AF
        DEC     A
        JR      NZ,DV0
```

You might wish to add comments to each statement.

**Experiment 8**

2.

| Hexadecimal | BCD |
|---|---|
| 0200H | 512 |
| FFFFH | 65535 |
| 18000H | 98304 |
| 5A48347FH | 1514681471 |
| 0100000000 | 4294967296 |
| 8000000000000000 | 09223372036854775808 |
| FFFFFFFFFFFFFFFF | 18446744073709551615 |

3.

```
        LD      (1801H),D
        LD      (1800H),E
        PUSH    DE
        LD      DE,0202H
        CALL    BINBCD
        LD      H,(1809H)
        LD      L,(1808H)
        POP     DE
```

4.

```
LD      A,E
ADD     A,A      *2
ADD     A,A      *4
ADD     A,A      *8
SUB     A,E      8-1=*7
```

4.

A second way

```
LD      A,E
ADD     A,A      *2
LD      (1A00H),A
ADD     A,A      *4
ADD     A,(1A00H)*4+*2=*6
ADD     A,E      *6+1=*7
```

**Experiment 9**

3.

The method shown below for BCD to binary conversion is a brute force method. Your solution may be different and shorter. The BCD number is contained in the HL register pair. The binary equivalent will be contained in the DE register pair. The B register is used as a loop count (16).

```
     H      L              D       E
 | BCD  number |      | Binary number |
```

```
1800                       ORG     1800H

                     ;CLEAR BINARY AREA AND CARRY FLAG

1800   1600                LD      D,0
1802   1E00                LD      E,0
1804   AF                  XOR     A

                     ;SET LOOP COUNT (SHIFTS)

1805   0610                LD      B,16

                     ;SHIFT H,L,D,E TOGETHER ONE BIT
                      TO THE RIGHT

1807   CB1C        SHIFT   RR      H
1809   CB1D                RR      L
180B   CB1A                RR      D
180D   CB1B                RR      E

                     ;ADJUST REGISTER H

180F   7C                  LD      A,H
1810   CB7F                BIT     7,A
1812   2802                JR      Z,COR1
1814   D630                SUB     30H
1816   CB5F        COR1    BIT     3,A
1818   2802                JR      Z,COR2
181A   D603                SUB     3
181C   67          COR2    LD      H,A

                     ;ADJUST REGISTER L

181D   7D                  LD      A,L
181E   CB7F                BIT     7,A
1820   2802                JR      Z,COR3
1822   D630                SUB     30H
1824   CB5F        COR3    BIT     3,A
1826   2802                JR      Z,COR4
1828   D603                SUB     3
182A   6F          COR4    LD      L,A

                     ;MORE SHIFTING? YES - JUMP BACK
                      TO SHIFT

182B   10DA                DJNZ    SHIFT

                     ;NO - SAVE DE AT 1A40-1A41

182D   21401A              LD      HL,1A40H
1830   73                  LD      (HL),E
1831   23                  INC     HL
1832   72                  LD      (HL),D
1833   FF                  RST     38H
```

# CHAPTER 7

## The Monitor

5 42

## Introduction

The purpose of a monitor is to allow the user to interface with the computer with a minimum of effort. At power up, the microprocessor monitor has to perform some initialization tasks--e.g.,check for location of the RAM memory, display uPF-I on the screen. The monitor then continously scans the keyboard, checking for a pressed key. If you press the key labeled PC, the monitor responds by accessing a routine labeled KPC (Key Program Counter). This routine will perform the series of tasks you would expect when pressing PC. The monitor has provided a human interface to the computer.

## WARNING

The monitor represent 2K of fairly advanced code. The explanation below is written for users who want an in-depth view of software. At times, you will have to struggle to understand the routines. Also answers are not provided to all the exercises. You should discuss your answers with associates or your instruction.

## 7 • 1 Major Divisions of the Monitor

Refer to the book MPF-I Monitor Program Source Listing. The listing is composed of two parts. The first part is the code. The code starts with statement 1 and ends at statement 2659. Find statement 2659 then turn to the next page. At the top of this page is the title Cross Reference. The cross reference listing is the second part of the listing. Look under Symbol for KPC. You sould see

| SYMBOL | VAL | M | DEFN | REFS |
|---|---|---|---|---|
| KPC | Ø1C2 | | 727 | 2434 2435 2435 2436 |

The column DEFN indicates (defines) at which line number the label (symbol) KPC is defined. Find line 727. The label KPC is on this line. The colon ":" forces KPC to be a label. KPC is a label for the code that follows. The code starts on line 731. Between line 731 and KPC are two comment lines and a blank line. The current value of the location counter for line 731 is given in the leftmost column. The value is Ø1C2. Look again at the cross reference listing under KPC. The column labeled VAL (value) contains Ø1C2. VAL is the first address of the routine KPC.

The REFS (reference) area tells the programmer all the statements that refer to KPC. The first reference is on line 2434. Find this line. Yes, KPC is referred to

```
2434   KFUN   DEFW   KPC
```

DEFW (define a word -2 bytes)-- tells the assembler to reserve space for two bytes and put the address of KPC in these bytes.

We will use the cross reference list again.

## 7 • 2 The Code

The first part of the monitor listing contains four major items:

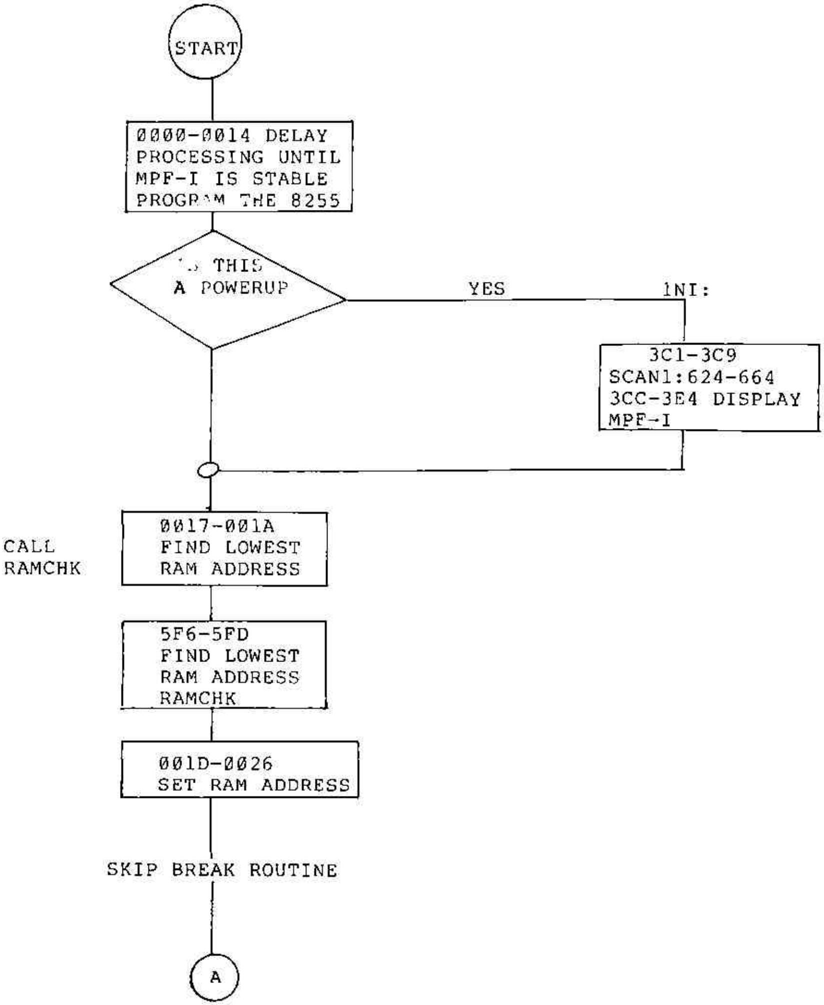
1) The start-up code.

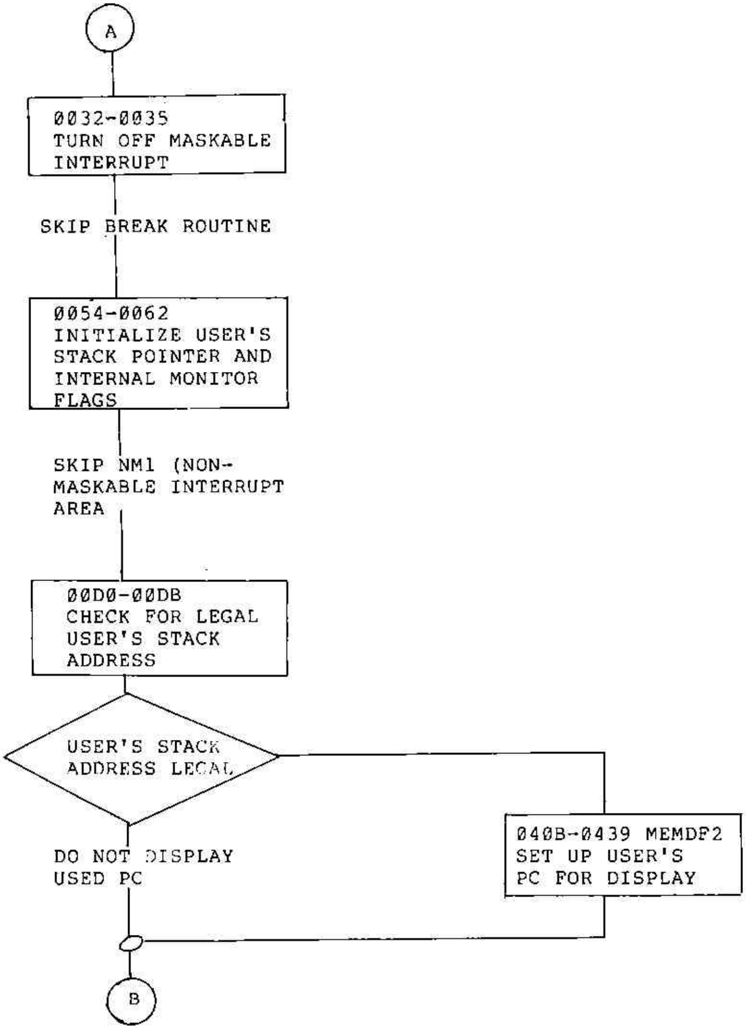
2) The routines which interrupt and respond to a key press.

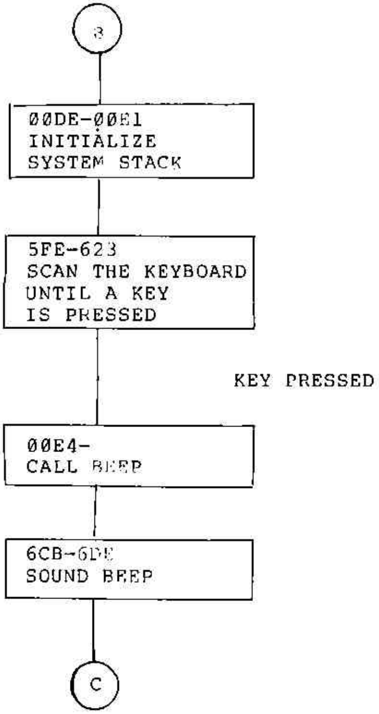
3) Special entry points, e.g., the interrupt INTR entry.

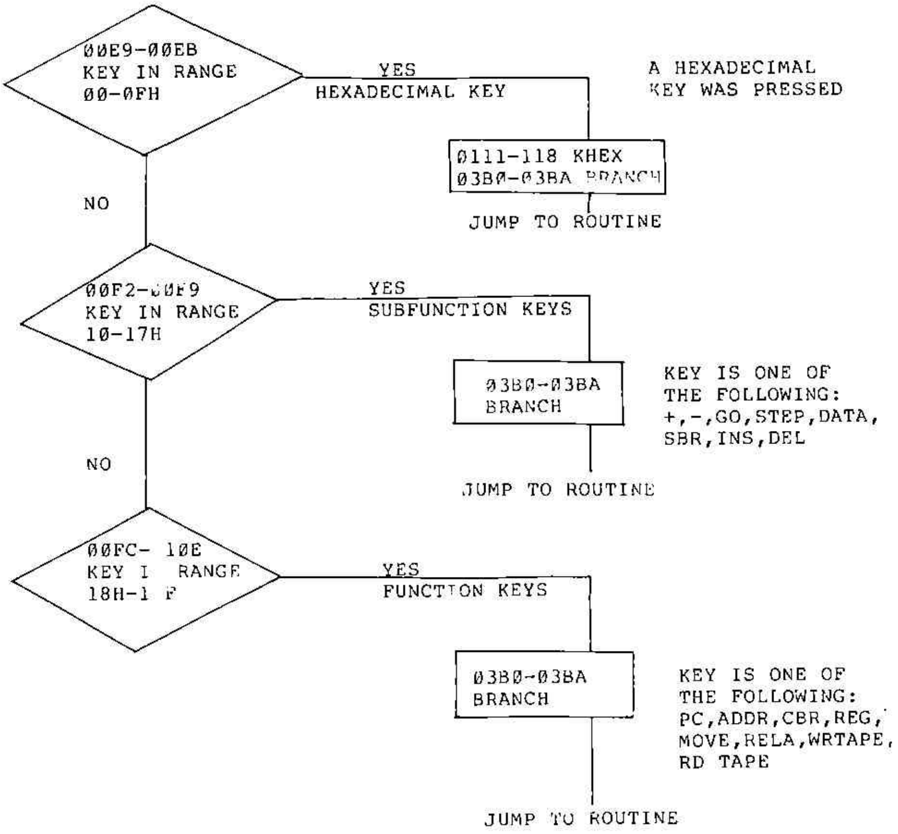
4) Tables

## AN OVERVIEW OF THE SEQUENCE OF ACTIONS AFTER THE POWER IS APPLIED




A
ØØ32-ØØ35
TURN OFF MASKABLE
INTERRUPT
SKIP BREAK ROUTINE
ØØ54-ØØ62
INITIALIZE USER'S
STACK POINTER AND
INTERNAL MONITOR
FLAGS
SKIP NM1 (NON-
MASKABLE INTERRUPT
AREA
ØØDØ-ØØDB
CHECK FOR LEGAL
USER'S STACK
ADDRESS
USER'S STACK
ADDRESS LEGAL
Ø4ØB-Ø439 MEMDF2
SET UP USER'S
PC FOR DISPLAY
DO NOT DISPLAY
USED PC
B


B
00DE-00E1
INITIALIZE
SYSTEM STACK
5FE-623
SCAN THE KEYBOARD
UNTIL A KEY
IS PRESSED
KEY PRESSED
00E4-
CALL BEEP
6CB-6DE
SOUND BEEP
C


ØØE9-ØØEB
KEY IN RANGE
ØØ-ØFH
YES
HEXADECIMAL KEY
A HEXADECIMAL
KEY WAS PRESSED
Ø111-118 KHEX
Ø3BØ-Ø3BA BRANCH
JUMP TO ROUTINE
NO
ØØF2-ØØF9
KEY IN RANGE
1Ø-17H
YES
SUBFUNCTION KEYS
Ø3BØ-Ø3BA
BRANCH
KEY IS ONE OF
THE FOLLOWING:
+,-,GO,STEP,DATA,
SBR,INS,DEL
JUMP TO ROUTINE
NO
ØØFC- 1ØE
KEY I RANGE
18H-1 F
YES
FUNCTION KEYS
Ø3BØ-Ø3BA
BRANCH
KEY IS ONE OF
THE FOLLOWING:
PC,ADDR,CBR,REG,
MOVE,RELA,WRTAPE,
RD TAPE
JUMP TO ROUTINE

## DETAILED EXPLANATION OF THE PRINCIPAL MONITOR FUNCTIONS

Use the flowchart 7-1 while tracing the monitor code.

Locations 0000--0003

Statements 100 & 101

When power is first applied to a circuit, the circuit should be allowed to stablize. Some I/O devices need a time delay before they can function. There are two time delays in the MPF-I. One time delay is in the hardware. The circuit connected to the Reset (RST) pin will prevent the MPF-I's Z80 from immediately starting execution. The other delay is provided by the two instructions at statements 100 and 101. B is first loaded with zero, then the DJNZ $ will decrement the value of B, and as long as B is non-zero, a jump to self (re-execute the instruction) will occur.

## Questions of Exercises

Exercise 7-1

How long is the delay at line 100?

Statements 106-107

These two statements program the 8255.

The 8255 chip has two lines A0 and A1 to tell it what function is being selected. If A0 and A1 are high, then the 8255 is being programmed. The instruction

```
        OUT (P8255),A
```

sends the controls in the A register over 8 address lines A0 to A7. P8255 is equal to 03 (hexdecimal) or 0000 0011 (binary). The right two bits which are high connect to A0 and A1.

Before the contents of the A register are output, it is loaded with a pattern, namely

```
1 0 0 1 0 0 0 0  Port C lower is output
   |  | | | |
   |  | | | └─Port B is output
   |  | | |
   |  | | └─Port B is mode 0
   |  | |
   |  | └─Port C upper is output
   |  |
   |  └─Port A is input
   |
   └─Select mode zero configuration for Port A
```

Mode zero means that the ports are used for input or output. Each half of Port C is programmed separately. Thus, Port A will accept input from the keyboard, the user key, and the cassette. Port B is used to control individual segments in a display. Port C is used to select a display, to scan the keyboard, and for output to the cassette, tone, and LEDs (light emitting diodes).

Statement 114 and 115

The output is to Port C of the 8255. This is Port Ø2 in MPF-I. The data sent to the 8255 will prevent a break. Bit 6 is made high. This sets PC6 (Port C, bit 6) high. The break circuitry is enabled by a low on Port C bit 6. The data sent to 8255 will also set the gate of the transister controlling the sound and the LED high.

Statement 116

The system stack pointer is set to an address in the RAM.

Exercise 7-2

What is the top of the system stack?

Use the cross reference to answer the question.

Statement 121-123

Read the contents of the location POWERUP. If it is not equal to PWCODE, then CALL subroutine INI. The designer assumed that on power up that the location PWCODE could not be equal to A5H--this is probably true.

Exercise 7-3

Using the cross reference list.

What is the address of POWERUP and INI?

What is the value in PWCODE?

Statement 1347-1368

The first pattern to be displayed will be all blanks. Look at the six bytes starting at location Ø7A5-- all zeroes. DEFB means define a byte. Register C will be used to make the uPF-I pass through the code at statement 1363 to 1368 seven times.

Statements 1363 to 1365 call SCAN1 ten times with IX register pointing to the same pattern. After the screen has been blanked out for Ø.16 second it is time to set up a new pattern. The next pattern is 5 leading blanks with a "u" in rightmost position. Decrementing IX, statement 1366 will give this pattern. Look at line 2536 to 2541. IX now points to location 7A4. The loop control statements DEC C and JR NZ,INI1 will decide whether to transfer control to statement 1363 label INI1. Control will be transferred to INI1 six times. At this point, the screen will read

MPF--I

Statement 1370 and 2382-2397

Load the PWCODE into A and then transfer control to INI3 (initialize code port 3). At INI3 statements 2382-2397 load the powerup code (byte) into location POWERUP, sets the beep frequency and duration, and returns to INI4. At INI4 statement 1372, a code of 0066 is loaded into HL. Then the next instruction puts this code into location IM1AD. Whenever a code of FF is executed, the MPF-I will transfer control to location 38H (effectively a call to 38H). The routine at location 38H will then direct the MPF-I to transfer control to IM1AD. Before the MPF-I goes to the code to establish, if a break point is in effect, the break point address is set to 0FFFFH. In the present MPF-I, address 0FFFFH does not exist. Read the comments at statements 1329 to 1383. At statement 1387, a return is made to location 0014 statement 123.

Exercise 7-4

Change the key beep frequency by trying different values in location 1FF1 statement 2657. Can You set the frequency so low that you can't hear it? Can the frequency be set so high that you cann't hear it?

Change the duration of key beep by changing locations 1FF2 and 1FF3. The monitor value is 2F in 1FF2 and 00 in 1FF3. Try larger values then smaller values. Try zero in 1FF2 and 1FF3. At least one value will cause you to loose control of the Micro-Professor. How can you regain control?

Exercise 7-5

(OPTIONAL-ANSWERS NOT PROVIDED)

Try to explain the comments and code at statements 194 to 214. You may need to read additional reference material. Make a drawing. It will help.

Statement 123-140 and 2103-2119

HL will point to location 1000H and then we are off to the subroutine RAMCHK at location 5F6 statement 2110. Does this routine look familiar? It should. The code was explained earlier. If location 1000H is the start of available RAM, then when RAMCHK is exited, the zero flag will be set. The return is location 1D statement 131. At this statement, we ask: Is the zero flag set? If yes, then transfer control to PREPC location 21 statement 133. The pointer to the beginning of user RAM (USERPC) will be loaded with 1000H. If a non-zero value is returned, then user RAM is assumed to start at 1800H. Statement 132 changes the value of the H register to 18. Before jumping to RESET1, register H is loaded with zero. Now H and L are both zero.

Statement 177-184

The interrupt register and the interrupt control flip- flop are not discussed in this manual. Consult the Z80 technical reference manual.

Statement 248-263

Statement 249 takes a fixed value USERSTK (1F9F) from ROM and loads it into the HL register pair. Then the next statement puts this value in USERSP. The contents of USERSP will point to the current top of the user's stack.

Exercise 7-6

Why is a pointer in RAM used to indicate the top of the user's stack?

Statement 251 and 252

The statements at 251 and 252 clear the byte labeled TEST. Bit zero must be set at the beginning of a new numeric entry. Setting bit zero of TEST to zero will automatically clear the data buffer when a hexadecimal key is pressed. The service routines for hexadecimal key entry reference routines are PRECL1 and PRECL2. (See statement 811-900). PRECL1 and PRECL2 test bit zero of TEST and preclears one (PREC1) or two (PRECL2) bytes (statements 1402-1428).

Setting bit 7 instructs the monitor to ignore the current key press and to send out a warning message. The routine IGNORE is called by keyboard routines which have discovered an illegal key press.

### A Case Study

You have pressed the set break point key SBR, and have entered an illegal address. The keyboard monitor routine will branch to routine KSBR (Key Set Break) statement 587 to 608. The KSBR routine uses RAMCHK to determine if your breakpoint address is a RAM location (CALL RAMCHK). If the address is not in RAM, a jump to IGNORE is executed (statement 1336). The routine IGNORE sets bit 7 as a warning message. The RET instruction transfers control back to the MAIN loop (statement 387). The next three instructions executed are

```
JR MAIN
LD SP,SYSSTK
CALL SCAN
```

SCAN tests bit 7 of TEST (statement 2138-39). If bit 7 is set, then the screen is blanked as warning of an illegal key press.

Did you enjoy tracking the effects of the flags in TEST. We got a little ahead of ourselves. The keyboard scanning routine has not yet been explained, but its kind of nice to get a preview. The actions of TEST may seem rather devious. One routine calls another routine which calls another routine. Some nesting of routines is permitted in a small monitor. In a large scale operating system, the accessing of nested routines must be carefully planned in advance Let us now finish the code RESET2.

Statement 258 sets IX to point to the initial display pattern MPF-I. When SCAN is called, IX is used as a display pointer(read statements 2125 to 2128) A jump to SETSTO avoids executing the code for a non-maskable interrupt.

Statement 353-373

Statements 360-361 clear the STATE. Read statements 459 to 474, you will gain some insight to the functions of the keyboard routines. The MPF-I uses a software breakpoint. A breakpoint is set by replacing the opcode of an instruction with a restart instruction RST 28H. A RST 28H saves the contents of the program counter--the next instruction to be executed--on the stack and then transfers control to the break routine at location 28H (statement 143). The replaced opcode is saved away in location BRDA (Breakpoint Data Address). The location of the breakpoint is BRAD (Breakpoint Address). Statement 362-363 will restore the data at BRDA to location BRAD. Why is this done? During power up statements 362 and 363 accomplish nothing because the breakpoint address is a nonexistant area of RAM. After power up, assume a program has been entered and a breakpoint set. The program starts executing and the breakpoint address is accessed. The program will halt. You now decide to investigate several registers and then to return to the monitor. Pressing MONI will transfer control to location 66H statement 266. The code at statement 266 to 351 is executed. At statement 351, a jump to BRRSTO statement 362 is performed. The breakpoint is removed. Thus by pressing MONI, you can return to the monitor and remove the breakpoint. Many monitors have this feature. The actions of the CALL C,MEMDP2 at statement 371 will depend upon which instruction sequence preceded this statement. If this is a power up sequence, then statement 258 sets IX to point to uPF-I and statement 360 cleared the carry flag. Control will not be transferred to MEMDP2 and MPF-I will be displayed. If the user's stack is not in RAM, then the code at statements 328 to 335 will display ERR-SP. If the user stack and the system stack use the same area (overlayed) then the code at statements 341 to 347 will display sys-sp. If no errors are detected after the MONI key is pressed, then the routine MEMDP2 is called. MEMDP2 (statements 1451-1492): 1) updates the state register, 2) calls routine ADDRDP to display the address of the program counter, 3) calls routine DATADP to display the contents of the address of the PC, 4) checks if the address to be displayed is a breakpoint, and 5) finally returns to location 00DE statement 380.

Statement 375-390

After setting the system stack, a call to SCAN will return the key code of the key pressed. The BEEP routine does the obvious thing--it sets up the parameter for a time and calls TONE to get a sound. Perhaps not so obvious is the command at statement 2411 JP KEYEXEC. When this command is executed, the A register contains the internal code of the key pressed. The routine KEYEXEC processes all keys except RS, MONI, INTR, and USER KEY.

As indicated earlier when a key function has completed, the RET in the key handling routine returns control to statement 387 (JR MAIN) which then re-execute MAIN.

Statement 392-457: KEYEXEC

KEYEXEC separates the internal codes of the keys into three groups. This is done to simplify the branching to each routine which process a key function. Read statements 2410 to 2422 to understand the branching method. The routine KHEX loads registers for use by the routine BRANCH. HL is loaded with a base address. Assume the GO key was pressed. The internal code for GO is 12. Since 12H is greater than 10H, statement 403 will not transfer control to KHEX. Statement 428 will subtract 10H, leaving the difference of 2 in A. 2 is less than 8, thus control is transferred to BRANCH--statement 1301. The object of BRANCH is to transfer control to the routine which will service the pressed key. A table has been designed to hold pointers (branch addresses) to the correct routine. The table for KSUBFUN is at statements 2424 to 2433. The function of BRANCH (in the case of GO) is to add together the contents of the first table address 011B (statement 2425) and the table entry for KGO, 0A (statement 2428). The address of the routine KGO is 123H (11BH + 0AH = 123H). Study statements 1301 to 1334 carefully to see how register A and register pair HL determine the jump address (statement 1333) by using the tables beginning at 2424.

Read the comments at statements 2411 to 2422, and 1302 to 1310.

This completes the explanation of the monitor. The interrupt system was not discussed.

Exercise 7-7

Tracing monitor code

Trace the actions of pressing a key from each of the three groups given under KEYEXEC. Verify your answer by reading the operations performed by the key in the User's Manual 3.1 Basic Operations.

If time permits, step through the code for all of the keys.

# CHAPTER 8

## How to Read a Schematic

When you read a schematic, you are looking at the results of the hardware design. A set of a hardware and software specifications are developed by a combined staff -- management, sales, software, and, of course, engineering.

When a microprocessor is composed of only a few chips, then a single sheet can show all of the schematics. The MPF-I contains 13 chips, a voltage regulator, displays, a keyboard, and two 40-pin extension connectors. Four sheets are required for the MPF-I schematics. Each sheet is numbered e.g. "Sheet 2 of 4"

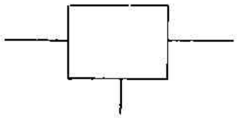
## Sheet 1 of 4

The components on this sheet, are vital to the MPF-I microcomputer. U1 is the Z80 CPU. U6, U7, and U8 are the memory chips, ROM and RAM.

We will first consider the requirements of the Z80 CPU.

## Voltage and Current

Most new CPUs use a single 5 volt voltage source. Chip specifications will tell the user (designer) the allowable voltage variations. Turn to sheet 4 of 4, you will find the voltage regulator. It is located in the upper right hand part of the page.



What voltage variations are allowed into the voltage regulator? (+7V to +24V) For the 7805 to work properly, the input, I, must be higher than the output, O. The output voltage is +5V.

The allowable input voltage range is given just to left of Input. What is it? (+7V +24V)

A smooth (clean) voltage can be supplied by the regulator, yet noise may appear on the voltage input to an IC. Some of the noise comes from circuits switching on and off. Each component that can malfunction due to noise (or sound spikes) must in some way be prevented from interacting with power supply. The property of a small capacitor is to allow high frequencies to pass through them. This action will filter out much of the noise. A noise spike is really a high frequency signal. Look immediately below the voltage regulator circuit and you will observe a series of capacitors shown as

These capacitors are typically used to filter out noise. Some filtering takes place before the voltage is regulated. The input capacitor has a capacitance of 4.7 uf. What is its part number? [C6]

Exercise

Look at your MPF-I board and find C1, C2....(drawing) If you were to design your own power supply, how could you learn about voltage regulators? Some manufacturers publish data books with explanations (tutorials) on how to use their components. An excellent book on voltage regulators is Motorala's Voltage Regulator Handbook. The author--Henry Wurgburg--has a section on "Selecting a Linear IC Voltage Regulator".

**Clocking Requirements**

The Z80 CPU can be operated over a range of clock frequencies.. The Z80 CPU in your MPF-I is certified by the manufacturer to operate at a maximum rate of 2.5 MHz. Designers sometimes specify a chip set allowing operation at a particular maximum frequency and then drive the chips at a lower frequency. There are two reasons for the lower frequency. One reason is that the circuits will operate more reliably. The other reason is that the clock may be performing another task requiring a specific frequency. Your MPF-I clock could be used to control the frequency (baud rate) of information sent and received in communications.

The clock circuitry (on sheet 1 of 4) is in the upper left hand corner (D-8). The base frequency of 3.58 megahertz is generated by a quartz crystal.

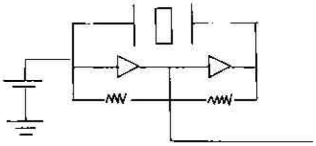
Exercise

The drawing for a crystal is two plates with the rectangular crystal drawn between the plates. Draw the crystal

[--|□|--].

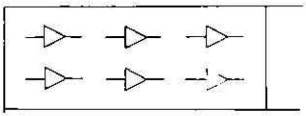
The crystal has the property of oscillating at an exact frequency when given a small amount of electrical energy.

The control circuit suppling the eletrical energy is composed of two sections of an IC, two resistors, and a capacitor.



In sheet 1 of 4, D-7

How can a circuit use only a few sections of an IC? It's simple. We can do this by only connecting two sections of the IC to the clock circuit.



The 74LS14 contains six elements sections called Schmitt Triggers (don't worry about detailed operation of a Schmitt Trigger). However, if you are interested in learning about Schmitt trigger, read the next section Schmitt trigger.

**Schmitt Trigger**

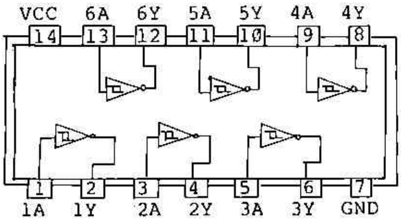
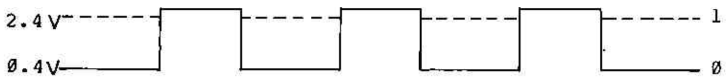
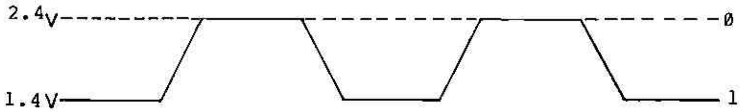



Fig. 74LS14 Hex Schmitt Triggers

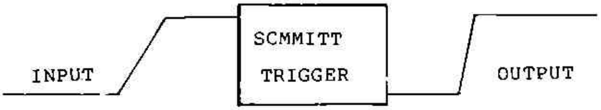
In digital circuits, the state of a signal can switch from Ø to 1 or from 1 to Ø. In conventional TTL circuits a zero centers at Ø.4V. Typically a value up to the Ø.8V and somewhat less than zero volts is accepted as a zero level. A one centers at 2.4V. Typically a value down to 2.ØV and all the up to about 5.25 volts is accepted as a one level. Here is an ideal TTL circuit.




Notice the immediate transition from Ø.4V to 2.4V. The transition can never be instantaneous but usually a quick transition is desirable. Here is a very slow transition.



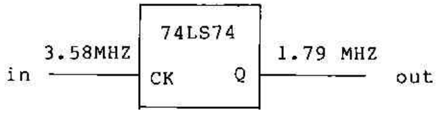

A slow transition can cause a problem. Devices attached to a slowly changing signal will become confused because too much time is spent in between the Ø and 1 level. The device is prone to say it's a one, no it's a zero, no it's a one. How can a quick rise time be achieved. A circuit called a Schmitt Trigger will wait until a changing voltage has passed a particular point and then snap to the new state.



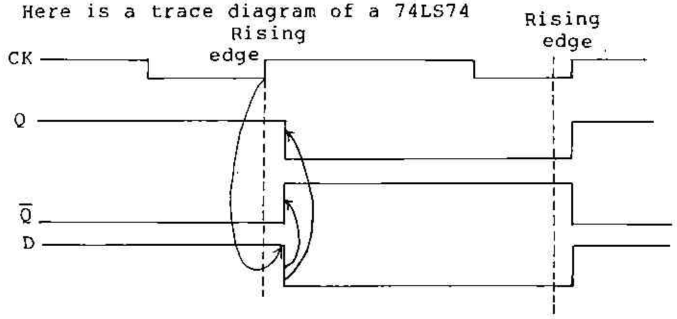
**DIVIDING THE OUTPUT OF THE CRYSTAL OSCILLATOR**

The crystal outputs a frequency of 3.58 MHz. But the Z80 CPU operates at 1.79 MHz. As you can see, the clock frequency was divided by two. Between the crystal circuit and the Z80 CPU is an IC, namely, 74LS74. The name of the function of a 74LS74 is Dual D Positive-Edge-Triggered Flip-Flops with Preset and Clear.

This information may not help your understanding of the circuit. Another way of looking at the function of the 74LS74 is shown below:



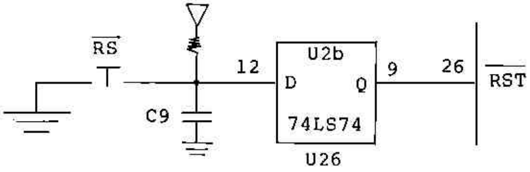

Here is a trace diagram of a 74LS74




Notice that the signal Q only changes on the rising edge of the CK (clock). Therefore, the clock is divided by two.

## Restart (RESET)

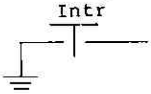
Your Z80 CPU may be out of control e.g. looping. A circuit controlling the RESET line into the Z80 CPU allows the operator to regain control. The object is to push a button and which will hold the RST (pin 26) line low for a few cycles. Then the line should go high.




Sheet 1 of 4, C-6

If RS is not pressed, pin 12 of 74LS74 U2b(D) is connected to 5V through a resistor. The 5 volts at pin 12 will pass through 74LS74 (U2b) so that pin 9(Q) will be high. Pin 9(Q) connects to the RST line and no action is initiated by the RST pin. Now press down on RS. Pin 12 will be grounded and a low will pass through U2b to pin 9. The low at pin 9 will present a low at pin 26 (the reset). This will cause the Z80 CPU to stop executing at its current address and immediately transfer control to location zero. You may wonder what the capacitor C9 does. It will hold the signal low for a few cycles when RS is pressed. Remember the RST line must be low for a few cycles.

There are two more ways of gaining control of the Z80 CPU. To be in control is to start at a predetermined address. Pressing the INTR key will transfer control to the monitor when the maskable interrupt system is enabled.

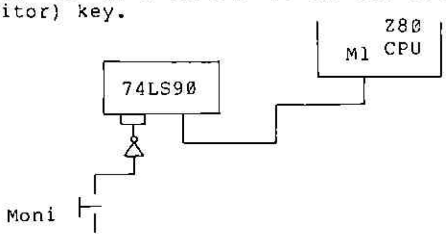



Pressing down on the INTR key will short the interrupt line ($\overline{INT}$) to ground. The $\overline{INT}$ interrupt is said to be active low. Signals which are active low have a bar above their names. When the INTR key is released the short to ground is removed and 5 volts is applied through resister R3. The resister which allows the voltage to be pulled up without damage to the power supply is called a pull up resister. Pulling $\overline{INT}$ low by shorting it to ground doesn't guarantee that the CPU will be interrupted. An $\overline{INT}$ will be ingored when the instruction DI (disable interrupt) has been executed. When the Z80 CPU is powered up, the maskable interrupt system is disabled.

The monitor code never enables the $\overline{\text{INT}}$ pin with an EI (enable interrupt) instruction. In the workbook, the uses of the maskable interrupt will not be discussed.

The second way to gain control of the Z80 CPU is to press the MONI (monitor) key.

Sheet 1 of 4, B-7




When the MONI key is pressed, a series of coordinated actions must occur. The object of pressing the MONI key is to cause a non-maskable interrupt (NMI). It is sufficient to know that a low at the pin marked with NMI will transfer control to memory location 66H. The coordinated actions are controlled by the 74LS90 which is a counter. The counter will change the level (e.g. high to low) of the signal at pin 17 ($\overline{\text{NMI}}$) of the Z80 CPU.

**Optional: A Detailed Analysis of the Operation of the 74LS90**

user presses down on the monitor key, a low must appear at the Z80 CPU's $\overline{\text{NMI}}$ with a minimum of delay. Secondly, when a break point is sensed during program execution, the break signal $\overline{\text{BREAK}}$ must be delayed until 4 instructions have been executed. The monitor key must take precedence over a break signal.

**The MONI Key**

Before MONI is pressed, a 5 volt level is applied to pin 9 of the 74LS14(d) coordinates B-7 on sheet 1. The 5 volts is supplied through the 10K ohm resistor. This resistor pulls the voltage level up when the MONI input is not grounded. The resistor is called a pull up resistor. When MONI is pressed, one end of the 10K ohm resistor is grounded and the level at 74LS14(d) goes to ground. The 74LS14 will invert the ground level from a low to a high. The level at pin 6 (R9(1)) and 7 (R9(2)) is high. Consult the 74LS90 truth table below

RESET/COUNT FUNCTION TABLE

| LINE | RESET INPUTS | | | | OUTPUT | | | |
|---|---|---|---|---|---|---|---|---|
| | RØ(1) | RØ(2) | R9(1) | R9(2) | $Q_D$ | $Q_C$ | $Q_B$ | $Q_A$ |
| 1 | H | H | L | X | L | L | L | L |
| 2 | H | H | X | L | L | L | L | L |
| 3 | X | X | H | H | H | L | L | H |
| 4 | X | L | X | L | COUNT | | | |
| 5 | L | X | L | X | COUNT | | | |
| 6 | L | X | X | L | COUNT | | | |
| 7 | X | L | L | X | COUNT | | | |

74LS9Ø -- Reset/Count Truth Table

Line 3 R9(1) and R9(2) high indicates that Qa will be high. Qa high will be inverted by the 74LS74(C) to a low. The low is presented to pin 17 $\overline{NMI}$ of the Z80 CPU. Pressing MONI interrupts the Z80 CPU. Line 3 also shows that the condition of RØ(1) and RØ(2) are irrelevant when R9(1) and R9(2) are high. This means that the MONI key takes precident over the BREAK.

**Break**

Understanding the actions of a breakpoint requires tracing both software and hardware. A breakpoint is set by replacing the opcode of the instruction at the selected breakpoint with a RST 28H instruction. When this instruction is executed, control will be transferred to location 28H. The routine to service an $\overline{NMI}$ (non-maskable interrupt) is located at 28H. The software sequence is 1) the routine KSBR (statement 587) responds to the user request for a breakpoint by setting the breakpoint address in location BRAD. 2) When the GO key is pressed, the service routine GDA (statement 1Ø24) puts the hex code EF (RST 28H) at the breakpoint address. When a break is recognized a transfer is made to the break trap routine at location 28H. The second part of the break routine starts at location 3E (statement 221). Statements 236 to 241 will now be analyzed in detail.

Statement 236  LD A,10000000H

The A register is loaded with the break enable pattern the leftmost bit is set.

Statement 237  OUT (DIGIT),A

The pattern in A is output to PC0 to PC7. PC7 sheet 2 of 4 coordinates B-4 connects to the 74LS90--sheet 1 coordinates B-7.

Statement 238 to 241

These four instructions will be executed before a non-maskable interrupt will occur. As long as both of the BREAK inputs Ro(1) and Ro(2) are high, the break will either set Qa through Qd low (see 74LS90 truth table lines 1 and 2) or have no effect if R9(1) and R9(2) are both high (line 3). Assume R9(1) and R9(2) are low and a $\overline{\text{BREAK}}$ signal is sent. The 74LS90 will begin to count. The count sequence is dependent upon how Ain and Bin are wired. The 74LS90 has QD connected to Ain. The count sequence is BI-QUINARY.

| COUNT | OUTPUT | | | |
|---|---|---|---|---|
| | $Q_A$ | $Q_D$ | $Q_C$ | $Q_B$ |
| 0 | L | L | L | L |
| 1 | L | L | L | H |
| 2 | L | L | H | L |
| 3 | L | L | H | H |
| 4 | L | H | L | L |
| 5 | H | L | L | L |
| 6 | H | L | L | L |
| 7 | H | L | H | L |
| 8 | H | L | H | H |
| 9 | H | H | L | L |

Count sequence for the 74LS90

The first four counts after the base value at COUNT 0 hold QA low. The fifth count COUNT 5 changes QA to a high and thus causes a non-maskable interrupt. A count occurs each time the line at Bin (pin 1) goes low. Bin is controlled by a signal ($\overline{\text{M1}}$) from the Z80 CPU which goes low every time a new instruction starts (or an extended opcode is read). Remember after the software issued the interrupt signal to the 74LS90 for more instructions were executed. Why was the 74LS90 choosen because both the MONI key and a BREAK could be serviced with MONI overriding BREAK. The counting feature of the 74LS90 may not be necessary.

## Memory Selection

The memory ranges of the ROM and RAM for the basic MPF-I are shown below:

| Address range in hex | Address range in binary | Chip functional/type |
|---|---|---|
| U6 0000--0FFF | 0000,0000,0000,0000--0000,1111,1111,1111 | Monitor/PROM |
| U8 1800--1FFF | 0001,1000,0000,0000--0001,1111,1111,1111 | Programs/RAM |
| U7 2000--2FFF | 0010,0000,0000,0000--0010,1111,1111,1111 | Programs/PROM |

Each binary bit is wired up to an address line. The address 19BF (hexadecimal)( = 0001 1001 1011 1111 binary) would be in the user RAM (U8). The corresponding address lines would be

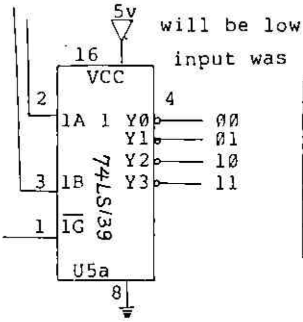
| A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

The upper four address lines A15 → A12 control the selection of which memory chip is active.

To select any of the memory chips A15 and A14 must be low. On the schematic sheet 1 of 4, (A, 5-4), there is a chip labeled 74LS139. Find U5a on sheet 1 of 4 (A, 5-4).

The 74LS139 is a "two to one of four" decoder. What this means is that if you enter one of four binary values 00, 01, 10, 11 into the chip, only one of the output lines is selected (goes low).

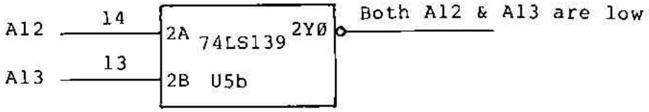
The 74LS139 has two sections. Each section has two input lines A and B, and a line which turns on (selects, enables) the section, and the four outputs.




| INPUTS | | | OUTPUTS | | | |
|---|---|---|---|---|---|---|
| ENABLE | SELECT | | | | | |
| G | B | A | Y0 | Y1 | Y2 | Y3 |
| H | X | X | H | H | H | H |
| L | L | L | L | H | H | H |
| L | L | H | H | L | H | H |
| L | H | L | H | H | L | H |
| L | H | H | H | H | H | L |

74LS139 Truth table

U5a is used to turn on (select) U6 or U7 or U8 only when 1A and 1B are low. 1A and 1B are connected to A14 and A15, respectively. So whenever A14 and A15 are both low, 1Y0 goes active low and one of the memory chips is activated. The pin labeled 1G which enables 74LS139(a) is active only when an instruction requesting memory is used. The instruction LD A,(HL) is a memory request instruction and will activate the memory request line $\overline{MREQ}$ pin 19 of the Z80 CPU (coordinates B-5). An instruction which will perform input-output operations such as IN A,(25H) will not activate $\overline{MREQ}$. Which memory chip is activated? To select U6 whose addresses range from 0000 through 0FFF, both A12 and A13 must be low.




Line 2Y0 in U5b goes low when both A12 and A13 are low. Therefore, 2Y0 of U5b is wired to the chip selection line of U6. A chip selection line activates a chip.

To select U7 address line A12 must be low and address line A13 must be high. What line of U5b should be selected? The answer is 2Y2.

Finally, to select U8, the following must occur: A13, low; A12, high; and A11, high. Do you see why A11 must be high? Because the memory chip on U8 has a range of addresses starting from 1800 (0001,1000,0000,0000) to 1FFF (0001,1111,1111,1111), any number which is smaller than 1FFF and bigger than 1800 is qualified to be used to point to a specific memory location. If you want to pick a binary number which is smaller than 1FFF and bigger than 1800, the conditions for such a number is that the 16th (A15), 15th (A14), and 14th (A13) bits should be 0 and 13th (A12) and 12th (A11) bits should be 1. That means A12 and A11 must be high.

Because A11 must be high, an additional decoder U9a is required. Trace the connections and see if you agree with the line selected to control the chip select (CS) of U8.

**Cross Reference of Sheet-to-Sheet Schematics**

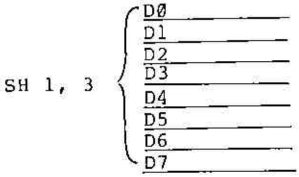
An inspection of the right or left margin of sheet 1 reveals some lines that do not connect to any components on sheet 1. However, the lines do have a label.

We will now formally discuss how to locate a specific location on the schematics and use cross reference between different sheets. Perhaps you have noticed that the four sides of a circuit map (schematic) are marked with A, B, C, and D, and 1, 2, 3, 4, 5, 6, 7, and 8. If we refer to the lowest and rightmost location of the circuit map, the words "(A-8)" is used to point to the location. Now find the location (D-1) on sheet 1. What you see is

```
D0--D7
----> SH2,3
```

The SH2,3 means that you shuld refer to the schematics on sheet 2 and 3.

The lines involved on (D-1) of sheet 1 are data lines--DØ through D7 (eight lines). Where do these lines go? SH2,3 indicates that sheets 2 and 3 are to be inspected. Turn to sheet 2 and use the coordinates (D-8) and (C-8), you will find:




The label SH1,3 indicates that lines DØ through D7 are connected to sheet 1 and sheet 3. At (B-1) is another set of lines connecting to sheet 2 and 3. Actually, only AØ, A1, A6, and A7 connect to sheet 2 and AØ and A1 connect to sheet 3. On the location (C-8) of sheet 1 is the RST line. It can also be found on (C-8) of sheet 2, and (A-8) of sheet 3.

**Extending the Capabilities of MPF-I**

All of the pins of the Z8Ø CPU are available as external signals. This feature allows all of the controlling signals available for use by add-on-boards. Turn to sheet 4 of 4, the pin assignment of the fourty pin connector P1 are shown.

## Sheet 2--The Control Function of the 8255

### The 8255

The dominant IC on sheet 2 of 4 is the 8255 (U16), which is installed on the location A-D, 5. This chip is designed to control input/output on three ports. When the MPF-I is turned on or reset, a monitor program determines how the ports of 8255 will function. Port A will be used for input and Ports B and C will always output. The 8255 competes with other chips for selection by the CPU. The decoder at (A-7) and (A-8) selects one of the three chips 8255, PIO, or CTC. The selected chip is said to be activated by the CPU. Each of these chips is said to be on a port. The details of I/O selection and control are not covered in this handbook. An extensive discussion of the actions of the displays U16 to U21 and the key matrix were covered earlier. However, the function of U12 and U15 (the 75491 chip) was not discussed. The 75491 is a segment and hex digit driver. The 8255 doesn't have enough power to drive displays. So an IC is required between the ports of the 8255 and the displays. U13 (the 75492 chip) is a driver which selects (activates) an entire display.

### Speaker

The speaker circuit at (B,C-1,2) of sheet 2 consists of a transistor Q2, resistor R9, and a speaker. The transistor is necessary to furnish more power than a typical integrated circuit can. Port PC7 of the 8255 controls the frequency and period of the sound.

### Cassette-Microphone

A resistance and capacitance at (B-1,2) shape the cassette recording signal output at port PC7.

### Cassette-Earphone

Diodes, a capacitor, a resistor, and two sections of the 74LS14 receive and shape the signal received from the cassette. This signal is then read into the 8255 at port PA7.

## User Key

The user key is a key that has no monitor functions and therefore is available for user definition. Sheet 3--Counter Timer Circuit (CTC) and Parallel I/Ø (PIO)

## Sheet 3--Counter Timer Circuit (CTC) and Parallel I/O (PIO)

The actions and programming of the CTC were covered in an earlier programming excercise. Parallel I/O using the Z8Ø PIO will not be covered in this course.

## Sheet 4

### P2--Pin Functions

You can locate the position of P2, which is a 4Ø-pin bus connector for the Z8Ø PIO and CTC.

The two ports of the Z8Ø PIO and the clock inputs and outputs of the Z8Ø CTC have been wired to the 4Ø-pin bus connector P2. The pin functions of P2 are listed on sheet 4.

### Memory Options

The user can install several different memory chips. This capability is made possible by allowing changes to the wiring. The user must cut some traces (wires) and jumper some points when using either a 2732 or 6116. A chart at (B-1,2,3) of sheet 4 shows the circuit changes.

# APPENDIX

## Appendix A Reference

### Z80 ONLY


1. Microprocessor Applications Reference Book
   Volume 1 00-2145-01, Zilog Inc.

2. Programming the Z80--Rodnay Zaks
   SYBEX

3. Z80 Assembly Language Programming
   Lance Leventhal, Osborne-McGraw Hill

4. Z80-Assembly Language Programming Manual
   Zilog 03-0002-01, Rev B. April 1980

5. Z80-CPU Z80A-CPU Technical Manual
   03-0029-01, Zilog Inc.

6. Z80 Microcomputer Handbook, William Barden
   SAMS

7. Z80 Microprocessor Programming and Interfacing
   Book 2, Nichols, Rony; Blackburg

8. Z80 Software Gourmet Guide and Cookbook
   Nat Wadsworth, SCELBI

9. Zilog 1981 Data Book


### COMPUTER CONCEPTS


1. Introduction to Microcomputers Volume 0
   (Basic Concepts) Adam Osborne
   Osborne-McGraw Hill

2. Introduction to Microcomputers Volume I
   (Basic Concepts) 2nd Edition
   Adam Osborne, Osborne-McGraw Hill

3. Introduction to Microcomputers Volume II
   (Some Real Microprocessors)
   Adam Osborne, Osborne-McGraw Hill

4. Introduction to Microcomputers Volume III
   (Some Real Support Devices)
   Adam Osborne, Osborne-McGraw Hill

5. Introduction to Microprocessors, Software,
   Hardware, Programming--Lance Leventhal
   Prentice Hall

6. Microprocessors and Programmed Logic
   Kenneth L. Short, Prentice Hall

## PROGRAMMING TECHNIQUES

(Not in Assembly Language)


1. PASCAL with Style, Henry F. Ledgard
   Hayden Book Company Inc., Rochelle Park,
   New Jersey

2. Programming Poverbs, Henry Ledgard,
   Hayden Book Company Inc., Rochelle Park,
   New Jersey


## MICROPROCESSOR DESIGN


1. Digital Hardware Design, John B. Peatman,
   McGraw Hill

2. Introduction to Microprocessor System Design,
   Harry Garland, McGraw Hill

3. Microcomputer-Based Design, John B. Peatman,
   McGraw Hill

4. Microprocessor System Design,
   Edwin E. Klingman, Prentice Hall


## INTERFACING


1. CMOS Cookbook, Don Lancaster, SAMS

2. Microcomputer Interfacing, Bruce Artwick, Prentice-Hall

3. TV Typewiter Cookbook,
   Don Lancaster, SAMS

4. Z80 Microprocessor Programming and
   Interfacing Book 2, Nichols and Rony,
   Blacksburg


## DATA COMMUNICATIONS


1. Data Communication and Teleprocessing Systems,
   Trevor Housley, Prentice Hall

2. Distributed Processing and Data Communications,
   Daniel R. McGlynn, Wiley Interscience

3. Technical Aspects of Data Communications,
DEC Educational Series JBØØ2A
Digital Equipment Corp.

## GENERAL REFERENCE

Computer Dictionary, Charles J. Sippl, SAMS

## Appendix B

### Alphabetical Listing of Monitor and Interrupt Key

The gray and orange topped keys are either sensed by a monitor keyboard scan routine or by a CPU interrupt. One exception is the user key--it is sensed by a user program.

| Name | Function | Reference |
|---|---|---|
| ADDR | Sets a memory address. | Page 11 |
| CBR | Clear the breakpoint in a user's program. | Page 23 |
| DATA | Inputs data either to memory or a register. | Page 11 |
| DEL | Deletes one byte from memory. | Page 27 |
| GO | Start execution at the current program counter address. | Page 18 |
| INS | Inserts one byte into memory. | Page 28 |
| INTR | Interrupts the executing program. This interrupt must be enabled by the user. | Appendix B sheet 1 of 4 |
| -(MINUS) | Decrements a value--use depends upon previous key function. | See Key functions |
| MONI | Interrupt the user's program. | Page 24 |
| MOVE | Moves a data block from one area to another area. | Page 25 |
| PC | Display current program counter--the STEP function will now be active. | Page 17 |
| +(PLUS) | Increments a value--use depends upon previous key function. | see Key functions |
| REG | Allows the user to select Z80 registers. | Page 14 |
| RELA | Computes the relative address in a jump relative instruction. | Page 29 |

| | | |
|---|---|---|
| RS | Performs a hardware reset. | Page 11 |
| SBR | Sets a breakpoint. | Page 20 |
| STEP | Executes a single step whenever pressed. | Page 19 |
| TAPE RD | Allows user to set up tape read parameters. | Page 32 |
| TAPE WR | Allows user to set up tape write parameters. | Page 30 |
| USER KEY | Available as a user programmed key. | Appendix B |

## Appendix C

### REGISTERS

The principal difference between a register and a memory location is the speed of accessing the data. The registers are on the CPU chip and can be accessed rapidly.

Memory access involves addressing memory and then fetching the data. The power of many CPUs is determined by the number and organization of the registers. The Z80 CPU has more registers than the 8080 or 8085-- two very popular chips. The Z80 CPU has many more registers than Motorola's 6800, but the register organization of the 6800 is different. To determine what registers are absolutely necessary in a computer which is register based. Read the explanation of registers below.

#### The A register

There must be an A register to hold the results of arithmetic (add & subtract) and logical (AND, OR etc.) operations. The register which participates in most arithmetic and logical operations is the A register --the accumulator. The register is a byte wide (8 bits) register. To add larger numbers than a byte's width, the A register is used repeatedly.

#### The HL register pair

Some method is needed to load numbers into the CPU registers so that sums, differences, etc. can be computed. The HL register pair is used frequently to point to a memory location. The instruction

```
LD A,(HL)
```

will load the A register with the one byte of data from the memory location pointed to by the HL register pair. Registers H and L may also be used separately (unpaired).

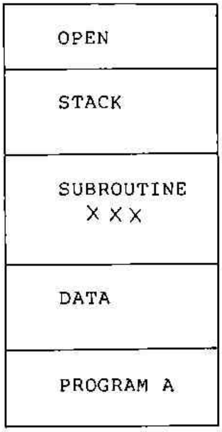
#### The BC and DE register pairs

Register pair BC and DE may also be used to point to memory. The register pairs can only be used to load the A register. The HL register pair will load a byte from memory into the A,B,C,D,E,H and L register. BC and DE are used in other instructions. For example, to perform a 16-bit addition. The instruction

```
ADD HL,BC
```

adds BC to HL. B,C,D and E may be used as 8 bit register (unpaired).

Consider the following arrangement of memory




Memory space management dirgram

**The PC register**

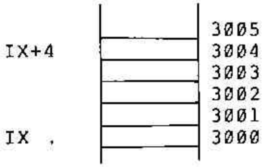
The processing begins by executing Program A. The location from which instructions are to be fetched (executed) is pointed to by a program counter, PC.

**The IX and IY register**

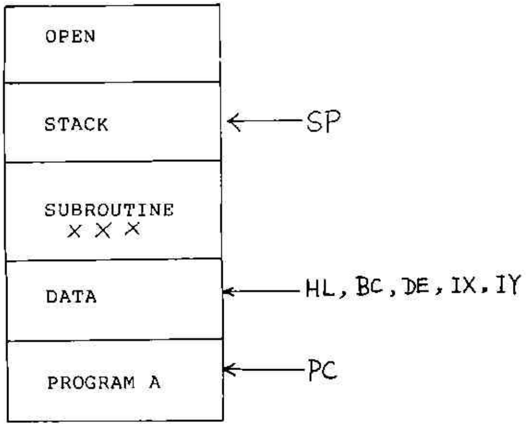
The data area can be accessed by any of the three pairs --HL, BC, and DE. H (high part of the address) and L (low part of the address) needed to be snapped together to form a 16-bit address. The Z8Ø possesses two more data pointers, the 16 bit registers IX and IY. pointers, the 16-bit registers IX and IY.
B and C, D and E, are also snapped together.
In many instructions, IX and IY, are used to point to a base address. The actual location accessed uses the base address plus an offset displacement. For example, the instruction

```
LD A,(IX+4)
```

loads the contents of the location from bytes beyond the value in the IX register.




In the diagram above where IX contains 3000, IX+4 would point to 3004. Because of the indexing feature (base address) of the IX and IY registers, the "I" stands for index. IX and IY are index registers. The Space Management Diagram can now indicates the usage of some registers.




**The SP register**

The stack is used to hold a temporary result or other non permanent information. When a transfer is made to subroutine XXX by Program A, the return address is stored on the stack. When the subroutine completes, the address on the stack is used to return control to program A. The stack is controlled by a stack pointer, SP. The stack pointer moves down or up depending upon whether data is added or removed. Turn to Appendix C in the MPF-I User's Manual. The second page of this appendix contains the Z80 CPU Register Configuration. Only the I and R 8-bit registers in the special purpose register area have not been mentioned.

### The I and R register

The I register is used with an Interrupt system designed to work with Z80. The R register supports a type of memory that needs a Refresh signal. Neither the I or R register are discussed in this workbook.

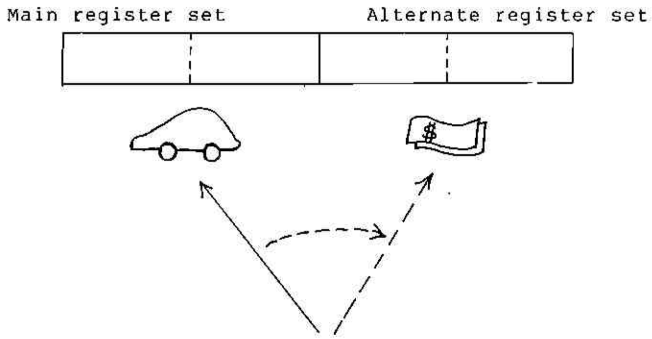
### The F register

The Flag register indicates the kind or type of result. After an ADD, was the result negative, zero, or positive? Did the ADD produce a carry? The appropriate flags and their meaning are described in the notebook chapters.

### The Alternate register set

The registers hold the current results and the next numbers to be processed. When you want to interrupt the current processing for a few seconds and service a short routine, you must typically preserve one or more registers. One way to preserve registers is to store them in memory. This storing process requires accessing memory--tell it to get ready and then moving the register contents from the CPU chip to memory. When the interrupt routine is run again, the reverse process must take place.

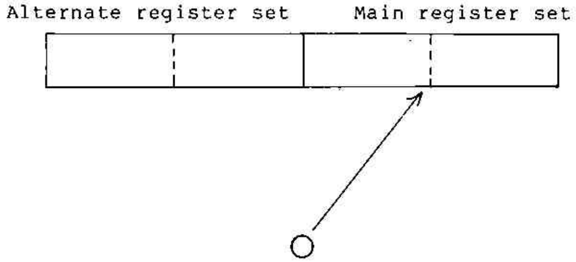
The Z80 CPU has a faster way to change executing routines. A computer much larger and more costly than the Z80 CPU was used a few years to process simultaneously (at the same time) two jobs (routines). The computer was bought to control traffic. In the morning the rush hour traffic would build up along certain avenues. Sensors placed on avenues and streets counted the number of vehicles passsing by. Periodically, the computer was interrupted to process the vehicle count. If a large number of vehicle were on a parti-cular avenue, then the signals were timed to move traffic faster on that avenue. When traffic processing was completed, the computer switched to processing the financial data for the city.




In the diagram above, the current processing is for traffic control. When the processing completes, the Z80 CPU can shift to the right set of registers by executing two instructions

EX AF, AF' and EXX

The alternate register set becomes the main register set.




The switch from one set of registers to another set will typically take from 2 to 4 millions of a second.

MULTITECH INDUSTRIAL CORPORATION
OFFICE/ 977 MIN SHEN E ROAD TAIPEI 105.
TAIWAN. R.O.C.
TEL:(02)769-1225(10 LINES)
TLX:23756 MULTIIC, 19162 MULTIIC.
FACTORY/5, TECHNOLOGY ROAD III 111
HSINCHU SCIENCE. BASED INDUSTRIAL PARK
HSINCHU TAIWAN. 300. R.O.C.
TEL:(035)775102(3 LINES)

Multitech Electronics Inc.

195 West El Camino Real
Sunnyvale, CA. 94086
U.S.A.
Tel: 408-7738400
Tlx: 176004 MAC SUVL
Fax: 408-7498032

DOC.NO:M1M09-302C